김일성종합대학출판사
주체91

확대병렬계산

(기술, 구성방식, 프로그람작성)

김일성종합대학출판사

차　　례

제2편. 허용기술

제4장. 제작블로크 : 극소형처리기

제5장. 분산기억기와 지연시간허용성

제6장. 체계호상접속과 기가비트망

제３편. 체계구성방식

제８장. 대칭 및 CC-NUMA 다중처리기

제４편. 병렬프로그람작성

제１２장. 병렬파라다임과 프로그람작성모형

제１３장. 통보문넘기기프로그람작성

제１４장. 자료병렬프로그람작성

머 리 글

병렬처리기는 콤퓨터설계의 전망목표이다. 기술의 끊임 없는 발전은 지금까지 병렬처리기를 만들어 내는 커다란 추동력으로 되여 왔다. 주요한 응용들에서 성능을 높이려는 요구도 역시 병렬처리기개발에서 중요한 역할을 놀았다. 그러나 효과적이며 확대가능한 병렬처리기를 만드는것은 지금까지 대단히 어려운 문제중의 하나로 되여 왔다.

대규모(n>100개처리기)병렬처리기개발은 많은 노력과 연구에 의하여 실현되리라고 보았으나 아직 완성되지 못하였다. 프로그람에서 필수적인 병렬성을 찾는것은 어려운것으로 되였고 이 병렬성이 발견되였을 때 그것은 포함된 처리기요소수에 비례하여 프로그람실행속도를 높이는데로 넘어 가지 못하였다. 이것은 특히 n이 100을 초과할 때 더욱 명백하다. 효률적인 병렬처리기를 실현하는데서 하나의 중요한 문제는 계산모형이다. 이 계산모형은 응용을 표현하는데 쓰이는 프로그람모형과 언어의 기초이다. 이 계산모형은 단일처리기를 위하여 개발되였으며 단일처리기에 아주 적합하다. 트랜잭션들에 대한 순차적규정의 언어적표시는 잘 알려 져 있기때문에 그것들은 직렬계산 및 프로그람작성모형에로 쉽게 넘어 간다. 이 표시들을 병렬모형에로 다시 넘기려는 시도는 지금까지 대단히 비효률적인것으로 되여 왔다.

효률적인 병렬처리에 대한 리해를 안받침하는 세개의 기둥 즉 계산모형, 기초적인 교체기술, 프로그람작성방식이 있다.

이 책은 이 세가지 문제에 대하여 유일하게 구체적으로 취급하고 있다. 성능에 대한 척도는 계산의 기초모형을 리해하기 위한 정량적기초이다. 이 책의 필자들은 하드웨어모형, 호상접속망 그리고 하드웨어정교성에 대한 완전한 표상을 주는 직렬광대역망을 포괄적으로 해석하였다. 그것들의 범위는 하드웨어 ILP(명령준위병렬성)에서부터 NOW(워크스테이션망)에 의하여 도달된 병렬성까지 포괄한다. 그 체계의 관점은 현재 진행중인 주요병렬처리기의 실현노력을 평가한데 기초하여 계산모형, 장치 그리고 프로그람작성모형을 통합하는것이다.

하드웨어와 쏘프트웨어의 결합에서 이 책은 대단히 가치가 있다. 응용에서 체계로 그리고 처리기구성에로의 넘기기를 리해함으로써만 효률적인 확대가능한 체계를 제작하는데서 일정한 진보를 기대할수 있다.

제1편. 확대가능성과 클라스터화

이 편에서는 병렬 및 분산형콤퓨터체계의 확대가능성과 프로그람가능성에 대한 원리들을 서술한다.

1장에서는 확대가능한 콤퓨터가동환경의 모형화에 대하여 서술한다. 확대가능성의 개념은 세개의 직교차원 즉 자원, 응용, 기술에 의하여 정의된다. 그다음 3개의 추상기계모형 PRAM, BSP와 위상병렬모형, 다섯개의 물리기계모형 PVP, SMP, MPP, COW와 MPP체계들의 특징을 서술한다. 또한 확대가능성에 대한 설계원리들을 실례에 기초하여 서술한다. 그 목적은 기술, 구조, 알고리듬, 언어, 응용, 사용되는 망환경에 무관계한 체계를 설계하기 위한것이다. 또한 균형설계, 여유설계, 역호환성에 대한 기본개념들이 서술된다.

2장에서는 특히 확대가능한 병렬콤퓨터의 프로그람화에 대하여 서술한다. 여기서는 프로쎄스와 과제, 스레드환경에 대한 기본개념외에 병렬성관리, 프로쎄스호상작용, 프로그람의미론, 알고리듬적형식, 쏘프트웨어이식성에서의 모든 중요한 문제들이 취급된다.

다음으로 병렬프로그람작성모형들을 소개한다. 모형의 세부에 대해서는 4편에서 취급한다.

- 병렬콤파일러모형
- 자료병렬모형
- 통보문넘기기모형
- 공유기억기모형

3장에서는 기본적인 성능평가기준과 척도에 대하여 취급한다. 그 목적은 확대가능한 성능에 대하여 속성들을 식별하기 위한것이다. 또한 병렬성능평가기준에 대하여 포괄적으로 소개한다. 다음으로 가격대성능비사이의 균형에 대하여 상세하게 고찰한다. 그리고 순차프로그람실행이 식별되며 병렬성관리에서 부가처리와 쏘프트웨어호상작용이 량적인 방식으로 해석된다.

또한 립도(granularity), 리용가능한 병렬성, 병렬성능척도, Amdahl의 법칙, Gustafson의 법칙 Sun과 Ni의 법칙, 여러가지 성능동형모형들이 성능평가기준결과들과 함께 정량적으로 해석된다.

제 1 장. 확대가능한 콤퓨터가동환경과 모형

이 장에서는 병렬콤퓨터들과 클라스터콤퓨터의 기본모형을 서술한다. 확대가능한 콤퓨터가동환경의 작용원리들과 기본설계문제들이 서술된다. 확대가능한 클라스터콤퓨터체계들은 주요한 구성방식특징을 가지고 모형화된다. 확대가능성은 3개의 직교차원 즉 자원, 응용, 기술에 의하여 실현된다.

1.3에서는 추상물리기계모형들의 특징을 서술하며 1.4에서는 다중콤퓨터클라스터화의 기본개념을 소개한다. 대칭다중처리기들, 콤퓨터클라스터들, 분산콤퓨터체계들사이의 차이도 고찰한다. 1.5에서는 확대가능한 병렬콤퓨터의 설계와 응용을 위한 기본원리들을 고찰한다.

bit, byte, 단어와 같은 단위들은 콤퓨터분야에서 널리 사용하지만 때때로 혼돈된 표시와 모호한 의미를 가지고 잘못 쓰일 때가 있다. 이 문제를 해결하기 위하여 우리는 이 책전반에 걸쳐 사용된 몇가지 표시법을 제시한다. 특히 독자들은 시간, byte, bit의 매개의 단위에 대한 간략표시를 혼돈하지 말아야 한다.

시간의 기본단위는 s로 표시된다. 두개의 기본정보단위는 byte와 bit이다. 1byte(1B)는 8bit(8b)이다. byte는 B로, bit는 b로 쓴다. 다른 정보단위는 단어(word)(16b 혹은 2B), 배단어(doubleword)(32b 혹은 4B), 4배단어(64b 또는 8B)이다. 이것은 Intel, Motorola, 수자장비에 사용되는 규칙에 기초한것이다.

한 단어를 32b로 하는데 어떤 초고속콤퓨터설계자들은 64b를 한 단어로 고찰한다. 자주 사용되는 작업부하단위는 류점연산의 수이며 flop로 간단히 표시한다. 계산속도의 단위는 초당 류점연산의 수이다(flop/s). 정보전송속도의 단위는 초당 byte수이다(B/s). 처리기의 실행속도는 흔히 초당 백만개 명령으로(MIPS) 하는데 유럽에서 사용되는 Mi/s표시와 같다.

표시법과 약속

우리는 표 1-1에서 규정된 약속과 표시를 쓴다. 10진과 2진해석은 고수준단위들에서 어느 정도 다르기때문에 독자들은 그런 차이들을 알아야 한다.

실례로 키로(kilo)는 10^3 =1000 또는 2^{10}=1024를 의미한다. 콤퓨터과학에서 K는 보통 1024를 의미하며 다른 과학분야에서 k는 1000을 표시한다. 일반적으로 2진해석은 bit, byte, 단어와 같은 정보단위에 적용된다. 10진해석은 흔히 실행시간, 전력, 무게, 작업부하계산 등에 쓰인다.

복수에 관한 약속은 다음과 같다. 단위는 전체 단어가 사용될 때 복수화되며 줄임표현이 쓰일 때에는 복수화되지 않는다. 실례로 2KB, 2kilobytes 또는 2Kbytes라고 말한다. 이것들은 모두 2048byte를 의미한다. 그러나 우리는 2KBs 또는 2Kbyte라고 쓰지 않는다. 2kb는 완전히 다른것 즉 2048bit 또는 256byte(256bytes)를 의미한다.

류사하게 2Mflop/s는 초당 2백만류점수연산을 실행하는 속도를 의미하는 표시이다. 2Mflops/s, 2MFLOPS 또는 2Mflops를 쓰는것을 피해야 한다. 이 책에서 로그함수는 언제나 밑수 2를 가진다. 즉 다른것이 규정되지 않는 한 log n은 $\log_2 n$을 의미한다.

표 1-1 수표시법과 약속

접 두 사	략 자	의 미	수 값
Milli	m	천분의 1	10^{-3}
Micro	μ	백만분의 1	10^{-6}
Nano	n	10억분의 1	10^{-9}
Pico	p	1조분의 1	10^{-12}
Femto	f	1000조분의 1	10^{-15}
Atta	a	100경분의 1	10^{-18}
Kilo	K(혹은 k)	천	10^{3} 혹은 2^{10}
Mega	M	백만	10^{6} 혹은 2^{20}
Grga	G	10억	10^{9} 혹은 2^{30}
Tera	T	조	10^{12} 혹은 2^{40}
Peta	P	1000조	10^{15} 혹은 2^{50}
Exa	E	100경	10^{18} 혹은 2^{60}

1. 1. 콤퓨터구성방식의 진화

1940년부터 지금까지 60년간 콤퓨터발전과정을 다섯개 세대로 나누어 고찰한다. 한 세대가 약 10~15년이다. 확대가능한 콤퓨터에 대한 형식적정의를 아래에 주고 그다음 미래의 병렬콤퓨터에 관한 구조를 서술한다.

1. 1. 1. 콤퓨터의 세대

콤퓨터기술은 지난 60년간 다섯개 세대에 걸쳐 발전하여 왔다. 표 1-2에서 기술, 구조, 쏘프트웨어, 응용, 대표적인 체계를 매개 콤퓨터세대에 따라 서술한다. 매개 세대는 하드웨어, 쏘프트웨어기술, 응용의 수준에서 이전 세대로부터 개선되여 왔다.

하드웨어의 분야에서 첫번째 세대는 진공관식계전기기억을 사용하였으며 두번째 세대는 개별 3극소자, 자심기억을 사용하였다. 세번째 세대는 소규모집적회로(SSI)를 쓰기 시작하였다. 네번째 세대는 초대규모집적회로(VLSI)극소형처리기를 쓰기 시작하였으며 다섯번째 세대는 극초대규모집적회로(ULSI)를 사용하였다. 실례로 1997년에 10M 3극소자 극소형처리기와 256M 3극소자 DRAM(동적자유호출기억)이 리용되였다.

쏘프트웨어분야에서 첫 세대는 기계/아쎔블리언어들을 사용하였고 두번째 세대는 고수준언어(HLL) 즉 Algol(알골)과 Fortran(포트란) 같은것을 쓰기 시작하였다. 세번째 세대는 C 언어다중프로그람작성, 시분할조작체계를 쓰기 시작하였으며 네번째 세대는 병렬벡토르처리를 중점으로 진행하였다. 다섯번째 세대는 이 책에서 취급된 확장가능하고 클라스터화된 계산을 기본으로 한다. 21세기에 콤퓨터의 다음세대들은 보다 더 발전될것이다.

표 1-2 50년동안의 콤퓨터세대

세대주기	기술과 구성방식	쏘프트웨어와 조작체계	대표적인 체계
첫번째 1946-1956	진공관과 계전기기억기, 축적기에 기초한 명령모임을 가진 단일비트CPU	기계/아쎔블리언어, 부분루틴이 없는 프로그람	ENIAC, IBM 701, Princeton IAS
두번째 1956-1967	개별 3극소자, 핵심기억기, 류동소수점축적기, I/O통로	콤파일러를 가진 Algol, Fortran, 묶음처리 OS	IBM 7030, CDC 1604, Univac LARC
세번째 1967-1978	직접회로, 관흐름된 CPU, 마이크로프로그람과 조종단	C언어, 다중프로그람작성, 시간공유 OS	POP-11 IBM 360/370, CDC 6600
네번째 1978-1989	VLSI극소형처리기, 반도체기억기, 다중처리기, 벡토르초고속콤퓨터	대칭다중처리, 병렬콤파일러, 통보문넘기기서고	IBM PC, VAX 9000, Cray X/MP
다섯번째 1990-현재	ULSI회로, 확대가능한 병렬콤퓨터, 워크스테이션클라스터, Intranet Inetrnet	Java, 극소핵심부, 다중스레드화분산OS, WWW	IBM SP2, SGI Origin 2000, Digital TruCluser

1. 1. 2. 확대가능한 콤퓨터구성방식

현대 콤퓨터의 특징은 병렬성이다. 단일처리기안에서조차 여러가지 방법으로 병렬성이 개척되여 왔다. 이것은 4장에서 자세하게 서술한다. 최근에 확대가능성의 개념은 다시 강조되였는데 그것은 병렬성을 포함하고 있다. 확대가능성개념은 이 책에서 중심으로 된다. 아래에서 확대가능성의 간단한 정의를 준다.

정의 1.1 콤퓨터체계가 그것의 하드웨어와 쏘프트웨어자원전체로 이루어 지면서 높아 지는 성능과 기능요구에 적합하게 확대할수 있거나(다시 말하면 자원을 증가한다.) 혹은 비용을 감소시키기 위해 축소할수 있다면(자원을 감소한다.) 콤퓨터체계는 확대(축소)가능하다고 말한다.

대부분 확대에 중점을 둔다고 해도 확대가능성이 커진다는것과는 동등하지 않다. 확대가능성은 축소하는 가능성을 동반한다. 확대문제는 한 처리기체계에서도 생긴다. 특히 체계가 확대가능하다는것은 다음과 같은 의미를 가지고 있다.

- **기능성과 성능** 확대된 체계는 보다 많은 기능성과 좋은 성능을 제공해야 한다. 체계의 전체 계산능력은 자원증가에 비례하여 높아 져야 한다. 체계자원이 n배 증가될 때 계산능력은 n배 높아 진다.
- **비용에 기초한 확대** 확대를 위해서 지불되는 비용은 적합해야 한다. 경험적으로

얻은 법칙에 의하면 *n*배로 확대하는데 *n* 또는 log *n*배보다 많지 않은 비용이 요구된다.

- **호환성** 하드웨어, 체계쏘프트웨어, 응용쏘프트웨어를 포함한 동일한 요소들은 약간 변화시켜 쓸수 있어야 한다.

사용자에게 전혀 새로운 조작체계를 주고 응용코드를 다시 개발할것을 기대하는것은 적합하지 않다. 확대는 체계의 일부분만을 포함한다. 실례로 보다 많은 처리기를 첨가하거나 처리기를 다음세대로 갱신한다. 이것은 체계의 다른 나머지부분들과 호환되여야 한다. 즉 이미 있던 기억기, 디스크, 호상접속주변련결은 여전히 쓸수 있어야 한다.

콤퓨터피라미드 확대가능체계에 대한 주요목적은 적응성이 있고 비용효과가 적은 정보처리도구를 마련하는데 있다. 그림 1-1의 실례에서처럼 콤퓨터들은 판매량과 성능, 비용에 따라 피라미드를 형성한다. 피라미드의 아래에 개인용콤퓨터(PC)부류가 있는데 그것들의 년간 판매량은 매우 높으며 성능은 제일 낮다.

초고속콤퓨터부류는 제일 우에 놓인다. 그것들의 판매량은 굉장한 가격으로 하여 제일 낮다. 그러나 초고속콤퓨터는 많은 자원들을 하나의 단일체계로 집적하고 최첨단기술을 가지고 있으며 성능이 가장 높다.

확대가능성개념은 모든 콤퓨터부류에 대하여 공통적인 확대가능구조를 가져야 한다. 이것은 다음과 같은 여러가지 우점을 가진다.

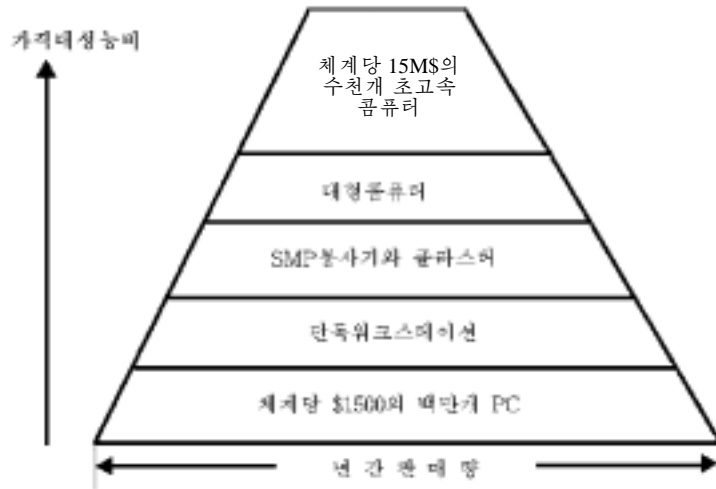



그림 1-1. 콤퓨터클라스의 피라미드

- 그것은 각이한 가격대성능비요구를 만족시킨다. 실례로 사용자는 낮은급체계를 구입할수 있다. 그것의 성능상 요구가 증가될 때 원래의 하드웨어와 쏘프트웨어는 여전히 새 체계에서 쓸수 있도록 체계를 확대할수 있다.
- 높은급기계들은 비용을 절약하기 위해 성능이 낮은 요소들을 쓸수 있다. 실례로 PC들은 낮은 원가, 쉽게 살수 있는 요소들을 가지고 있다. 확대가능한 구조를 가

지고 있는것으로 하여 초고속콤퓨터들은 원가를 낮추기 위해서 이러한 요소들을 쓸수 있다. 필수적요소들(처리기, 기억소편, 디스크, I/O조종기 등)을 쓰는것은 고성능체계개발의 하나의 추세로 되였다.

- 높은급체계를 위하여 개발된 첨단기술은 결국 저성능체계의 성능을 개선하며 비용효과성을 높일수 있다.

1．1．3．체계구성방식의 수렴

확대가능한 병렬콤퓨터들은 세가지 구성방식에로 수렴하고 있으며 그것들은 그림 1-2에서 보는바와 같이 자원공유의 수준증가와 관련한 많은 공통성을 가진다.

공유를 가지지 않는 구성방식은 호상 접속망으로 련결되는 여러개의 마디들로 구성된다. 마디는 보통 쉘구성방식에 따르며 여기서 하나의 전용으로 설계된(쉘이라고 부르는) 회로장치는 극소형처리기를 마디의 나머지부분 즉 캐쉬, 국부기억기, 망대면부회로장치(NIC), 디스크에 련결된다.

디스크공유구조는 디스크모듈이 마디들사이에서 공유되도록 마디밖으로 옮겨 진다는 데서 공유구조와 다르다. 기억기공유구조에서는 지어 주기억까지 공유된다.

마크로구성방식과 마이크로구성방식 콤퓨터체계의 전체 구조를(실례로 그림 1-2) **마크**

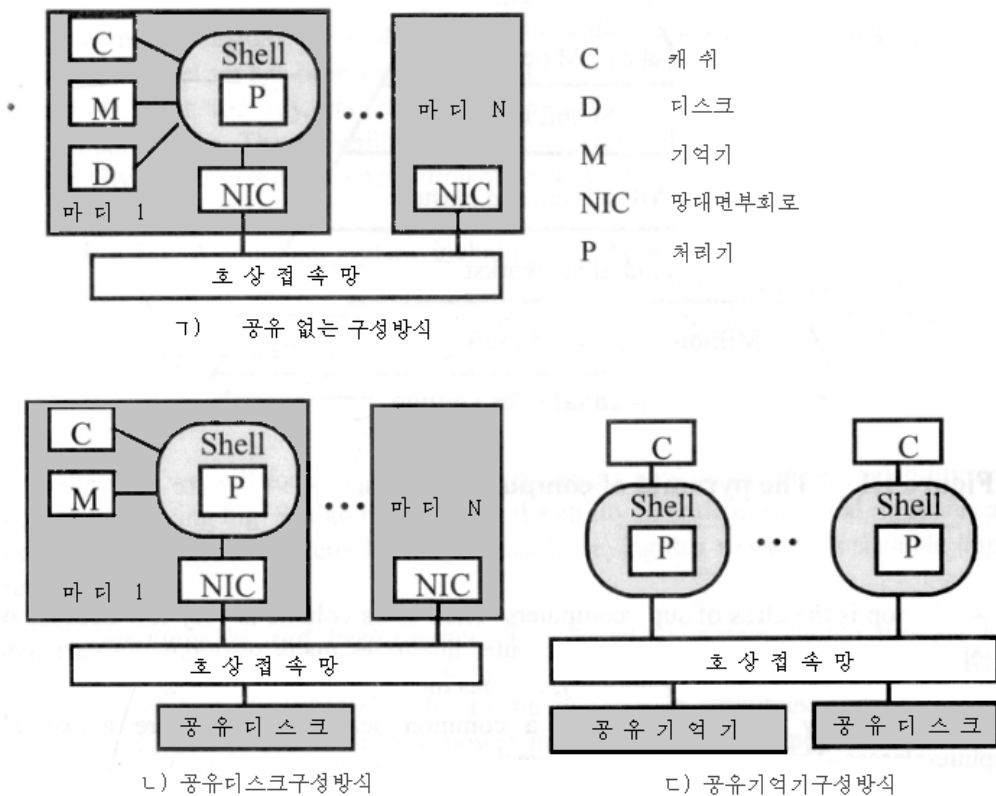



그림 1-2. 확대가능한 병렬콤퓨터에 대한 일반구성방식

로구성방식이라고 부르며 처리기와 그것을 둘러 싼 쉘을 **마이크로구성방식**이라고 부른다. 쉘구조의 우점은 처리기가 다음세대로 갱신되거나 다른 구성방식으로 변경될 때 오직 쉘(혹은 마이크로구성방식)만이 변경될것을 요구한다는것이다.

우의 모든 구조적개념은 실제병렬콤퓨터들을 간단화한것이다. 실례로 공유기억기, 공유디스크는 공유가 아닌 구조의 호상접속에 첨가될수 있다(그림 1-2 ㄱ). 이것은 그림 1-2 ㄴ, ㄷ)에서 보는것처럼 공유디스크 혹은 공유기억을 가진 구조로 될수 있다. 자세한 설명은 뒤장들에서 취급한다.

1. 2. 확대가능성의 차원

확대(축소)(체계자원을 증가시키거나 감소시키는)에는 여러가지 방법이 있다. 그것들은 확대(축소)가능성의 각이한 관점과 정의에로 귀착된다[69, 513]. 앞으로 우리는 확대화에 대하여 중심적으로 고찰한다.

처리기는 CPU를 가리키는 말이다. CPU는 하나 또는 그이상의 명령관흐름, 단, 캐쉬 기타 다른것들을 포함한다. 처리기는 단일소편우에서 실현되거나 또는 단일다중소편모듈우에 설치된 다중소편들우에서 실현될수 있으며 그렇지 않으면 다중분리소편들로써 실현될수 있다. 마디는 하나이상의 처리기들, 국부기억기, 다른 회로장치들을 가진다. 간단히 우리는 따로 규정하지 않는 한 그림 1-2에서 보여 준 모형에서의 마디들을 단일처리기로 가정한다.

1. 2. 1. 자원확대가능성

자원확대가능성이란 기계크기(다시 말하면 처리기의 수)를 늘이고 보다 많은 기억(캐쉬, 주기억기, 디스크)을 장비하며 쏘프트웨어를 개선함으로써 보다 높은 성능을 얻는다는것을 말한다.

크기확대가능성 콤퓨터체계를 확대하는 가장 명백한 방도는 기계크기 다시 말하면 처리기의 수를 늘이는것이다. 크기확대가능성은 체계가 적합하게 쓸수 있는 처리기의 최대수로 평가된다. 모든 병렬콤퓨터들이 동일하게 크기확대가능한것은 아니다. 실례로 1997년 현재 대칭다중처리기(SMP)체계는 약 64개 처리기까지 확대할수 있고 한편 IBM SP2은 512개 처리기까지 확대할수 있었다.

현재 병렬콤퓨터에서 크기를 확대할 때 처리기를 첨가하는것은 간단하지 않을수 있다. 통신부분체계들은 호상접속, 대면부, 통신쏘프트웨어를 포함하며 이것들은 역시 개선되여야 한다. 보다 큰 하드웨어병렬성을 어떻게 실현하는가 다시 말하여 확대체계에 대한 계획을 어떻게 세우겠는가 하는 문제가 제기된다. 병렬체계의 확대가능성을 제약하는 두가지 주요인자로서 프로그람작성과 통신이 있다. 이 문제는 뒤장들에서 자세히 취급한다. 현존상업적인 연구체계들에서 어떻게 그 문제들을 효과적으로 해결하는가를 배우게 될것이다.

자원의 확대 처리기를 첨부하는것은 체계를 확대하는 가장 명확한 방도이기는 하지

만 이것이 유일한 방도는 아니다. 같은 수의 처리기를 가지고 있지만 더 많은 기억, 더 큰 소편외장캐쉬, 더 큰 디스크와 기타 여러가지를 장비하여 쓴다.

실례 1.1. IBM SP2에서 기억요구

Maui고성능계산쎈터(MHPCC)가 400마디 SP2체계를 갱신하기로 하였는데 더 많은 마디를 첨부하는 대신에 디스크용량과 기억은 증가하기로 하였다. 확대된 기억용량은 표 1-3에서 보여 주었다. 이것들은 실례로 그룹 1에 8개의 Wide형마디들이 있고 매개는 1GB의 주기억, 4.5GB의 국부디스크를 가진다는것을 보여 준다. 여기서 1000MB는 주기억과 보조기억사이의 교체를 위한 페지화공간으로 쓰이며 또 다른 2000MB는 국부작업공간(scrstch)으로 사용된다.

표 1-3 MHPCC에서 IBM SP2의 기억기와 디스크용량

그룹	마디형태	마디수	기억기 (MB)	디스크 (GB)	페지화 (MB)	국부작업공간 (MB)
1	Wide	8	1024	4.5	1000	2000
2	Wide	56	256	4.5	500	2000
3	Wide	16	256	2.0	500	1000
4	Thin	160	128	2.0	256	1000
5	Thin	160	64	1.0	180	250

기억기공간을 확대하는것은 많은 기억기소편을 사는것보다 더 리롭다.

체계는 확장된 기억을 쓸수 있게 기억기용량을 설계하여야 한다. 실제적인 체계는 언제나 최대기억용량의 웃한계를 가지고 있다. 실례로 IBM SP2은 마디당 최대한 2GB의 기억을 쓸수 있으며 Cray T3D는 오직 64MB만을 쓸수 있다. 이 용량들은 주로 비용과 기술의 관계에 의하여 제한된다.

쏘프트웨어확대가능성 확대가능콤퓨터체계의 쏘프트웨어는 아래에서 보여 주는것과 같은 방도에 의하여 개선될수 있다.

조작체계의 보다 더 새로운 판본, 그것은 다중스레드처리와 같은 더 많은 기능을 가지며 많은 사용자처리와 주소화공간, 더 효과적인 핵심부기능, 기타 등을 지원한다.

- 보다 효과적인 최량화기능을 가진 좋은 콤파일러
- 보다 효과적인 수학적 및 공학적서고들
- 보다 효과적이고 쓰기 쉬운 응용쏘프트웨어
- 리용자에게 편리한 프로그람작성환경

1. 2. 2. 응용확대가능성

확대가능한 병렬콤퓨터의 능력을 충분히 리용하기 위하여서는 응용프로그람들이 확대가능하여야 한다. 이것은 같은 프로그람은 확대된 체계에 따라서 더 좋은 성능으로 실행되여야 한다는것을 의미한다. 두가지로 구분하면 기계크기의 확대가능성과 문제크기의 확대가능성이다.

기계크기의 확대가능성 이것은 처리기가 보충될 때 성능이 얼마나 더 개선되는가를 나타낸다. 실례로 n-처리기콤퓨터체계가 자료기지봉사기로 사용된다고 하자. 그것은 인구자료기지를 가지고 있으며 보통 100명의 과학자들이 질문을 제기하면 초당 1000개 트랜잭션을 처리하는 성능(TPS)을 가지고 있다.

처리기의 수를 배로 하여 $2n$으로 만들면 어느 정도의 속도로 개선될수 있겠는가?

체계의 속도를 2000TPS로 증가시키면 체계는 기계크기를 초월하여 확대된다고 말할수 있다. 증가되는 자원이란 흔히 처리기들이지만 그것들은 역시 기억공간 또는 I/O가능성이라는것을 알아야 한다.

문제크기의 확대가능성 이것은 체계가 보다 큰 자료크기, 작업부하를 가지는 문제를 얼마나 잘 조종하는가를 가리킨다. 자료기지봉사기의 실례를 다시 고찰하자. 봉사기가 중국의 인구자료기지를 가지고 있다고 하자. 봉사기는 얼마나 잘 동작하겠는가? 자료기지의 크기는 5배로 증가되였으며 사용자수가 200으로 증가되였다면 무엇이 일어날것인가?

응용의 확대가능성을 연구할 때 다음의 세가지 문제점이 제기된다.

- 많은 실제적인 병렬응용들은 기계크기와 문제크기에 따라서 한계를 설정하였다. 실례로 병렬탐지기신호처리프로그람은 최대로 256개의 처리기를 쓰며 최대로 100개의 탐지기통로를 조종한다. 이 한계는 단순히 기계자원을 증가시키는것에 의하여 초과될수 없다. 프로그람은 보다 많은 처리기와 탐지기통로를 조종할수 있게 변경시켜야 한다.
- 응용은 규정된 기계와 결부시켜 고찰해야 한다. 이 응용/기계쌍을 때때로 하나의 체계로 본다.
- 응용의 확대가능성은 반드시 기계크기와 문제크기에 의존하지 않는다. 그것은 역시 기억기용량, I/O능력, 기계의 통신능력에 의존한다. 이 모든 인자들은 확대가능성에 밀접하게 영향을 준다. 우리는 이 문제들을 3장에서 취급할것이다.

1. 2. 3. 기술확대가능성

기술의 확대가능성은 기술적인 변화에 대처할수 있는 확대가능한 체계에 적응된다.

그것은 세가지 부류 즉 세대확대가능성, 공간확대가능성, 이종확대가능성으로 갈라질수 있다.

세대(시간)확대가능성 체계는 빠른 처리기, 빠른 기억기, 더 새로운 조작체계판본, 보다 위력한 콤파일러 등과 같은 다음세대구성요소를 리용하여 확대할수 있다. 또한 계산능력은 체계가 다음세대로 넘어 갈 때 증가되여야 한다. 게다가 체계의 나머지부분은 될수록 적게 변경시켜 쓸수 있어야 한다. 사용자가 처리기나 조작체계를 한층 높이 갱신할 때 전체 체계를 다시 장비하게 하는것은 적합치 않다. 근사적으로 콤퓨터체계에서 가장 빨리 발전하는 요소는 처리기인데 그것은 18개월 또는 2년마다 그것의 성능이 두배로 증가된다. 가장 뜨게 발전하는것은 프로그람작성언어들(Fortran 77은 여전히 널리 쓰인다.)인데 아주 능력이 높은 콤파일러라도 수년간마다 개발된다. 성능에 대한 관심이 있는 사용자는 콤퓨터체계의 처리기는 2년마다 갱신하고 나머지요소들은 훨씬 더 낮은 비률로 갱신되기를 원할수 있다. 유감이지만 이것은 과거의 병렬콤퓨터들에서는 생각할수 없었고 그때 매개 세대는 사용자응용의 재개발과 체계쏘프트웨어에 대한 새로운 투자를 요구하였다.

실례 1.2. IBM개인용콤퓨터의 세대확대가능성

가장 좋은 세대의 확대가능한 콤퓨터는 아마도 IBM PC일것이다. 많은 PC사용자들은 자주 처리기, RAM용량, 다매체확장과 같은 체계요소들이 자주 갱신되여 계산능력은 증가되나 체계의 기타부분은 그대로 리용할수 있다는것을 알게 되였다. 나머지체계는 현시장치, 하드구동기, 인쇄기 등을 포함한다.

특히 중요한것은 그들이 현재체계와 응용프로그람(DOS, Windows, 자료기지, 스프레드쉬드, 문서처리쏘프트웨어, 기타 등)의 2진코드는 변화시키지 않고 갱신된 체계우에서 더 빨리 실행할수 있다는것을 알수 있다. 왜냐하면 PC체계(처리기로부터 시작하여 모판, I/O카드, 쏘프트웨어에 이르기까지)는 세대확대가능하도록 설계되였기때문이다.

공간확대가능성 이것은 체계의 능력을 한개 함, 방, 건물에 있는 다중처리기로부터 여러 건물과 지리적범위에로 규모를 확대하는 체계의 능력을 가리킨다. 이런 견지에서 SMP와 MPP는 제한된 공간확대가능성을 가지며 한편 인터네트는 우월한 공간확대가능성을 가진다.

이종확대가능성 이 성질은 체계가 각이한 설계자들 혹은 판매자들이 공급하는 하드웨어와 쏘프트웨어요소들을 집적하여 얼마나 잘 확대할수 있는가를 가리킨다. 이것은 표준이고 열린 구조와 대면부를 가지는 요소들을 사용할것을 요구한다. 쏘프트웨어부분에서 이것을 **이식성**이라고 부른다.

실례 1.3. 확대가능한 병렬콤퓨터에서 쏘프트웨어이식성

IBM병렬조작환경(POE)은 RS 6000체계의 임의의 크기이상으로 확대할수 있다. POE를 쓰면 병렬프로그람 RS6000마디들로 된 망에서 변경없이 실행될수 있는데 여기서 매 마디는 낮은급 PowerPC워크스테이션 혹은 높은급의 SP2wide마디이다.

이 마디들은 임의의 통용되는 호상접속에 의해서 련결될수 있는데 그것은 느린 Ethernet로부터 SP2의 고성능교환(HPS)에 이르기까지 포함한다. 이 마디들은 지형학적으

로 떨어 져 있는 위치에 있을수 있는데 하나는 남부캘리포니아에 있고 나머지는 Maui고성능계산쎈터에 있다.

또 하나의 실례는 병렬가상기계(PVM)인데 그것은 역시 이종확대가능하다. 그것은 병렬프로그람이 서로 다른 판매자로부터 구입한 마디들로 이루어 진 망우에서 실행될수 있다.

1. 3. 병렬콤퓨터모형

병렬기계모형은(프로그람작성모형, 타이프구조, 개념모형, 리상모형으로도 알려 져 있다.) 프로그람작성자의 견지에서 보면 하나의 추상병렬콤퓨터이며 그것은 순차계산을 수행하는 von Neumann모형과 류사하다. Purdue Reort[556]에 의하면 이러한 모형은 병렬계산에서 기본인 병렬콤퓨터의 능력을 특징 지어야 한다. 그러나 특수화된 병렬콤퓨터들은 그것들을 제공한다고 볼수 있다.

추상화는 처리기의 수, 처리기사이 통신구조와 같은 구조적정보를 포함하지 말아야 하지만 암시적으로는 병렬계산의 상대적비용을 구할수 있어야 한다. 이러한 모형은 실현세부에 대하여서는 명백하지 않아도 성능에 대하여서는 충분히 정확해야 한다. 이러한 추상모형은 콤퓨터구조설계가들, 쏘프트웨어개발자들, 프로그람작성자들, 알고리듬설계자들에게 많은 리득을 준다. 매 병렬콤퓨터는 자기의 구조를 밀접하게 반영하는 순수한 모형을 가진다.

1. 3. 1. 의 미 속 성

병렬기계모형의 특징은 다섯개의 의미속성과 여러개의 성능속성에 의하여 묘사될수 있다. 이 속성들은 리론적인 기계모형을 보면 리해될수 있다. 크기 *n*의 병렬자유호출기계(PRAM)는 그림 1-3에서 보는것처럼 *n*개의 처리기로 구성되며 모두 하나의 공유기들에 접근한다.

PRAM우에서 병렬프로그람은 *n*개의 처리기로 이루어 지며 *i*번째 프로쎄스는 *i*번째 처리기안에서 실행되고 명령렬로 구성된다. 매개 처리기는 매 기본시간단계(cycle이라고 부른다.)에서 하나의 명령을 수행한다. 전형적인 순차형콤퓨터에서 볼수 있는바와 같이 명령은 자료전송, 산수/론리, 조종흐름, I/O명령들을 포함한다.

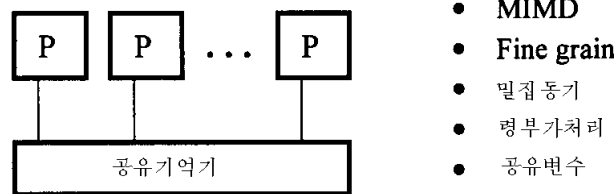



그림 1-3. PRAM(병렬자유호출기계)모형

동형성 이 속성은 병렬콤퓨터의 처리기들이 병렬프로그람을 실행할 때 얼마나 류사

하게 동작하는가 하는 특징을 의미한다. 크기가 1인 PRAM은 일반적인 자유호출기계[10]로 되며 여기서 프로그람은 단일한 명령흐름을 실행하고 단일한 자료흐름에 접근한다. Flynn의 분류[244]에 따르면 이러한 기계는 단일명령(흐름), 단일자료(흐름)(SISD)기계이며 그것은 보통의 순차형콤퓨터를 모형화한다.

하나이상의 처리기가 있을 때 PRAM은 다중자료흐름에 접근하여 다중명령흐름을 실행한다. 따라서 PRAM은 다중명령(흐름), 다중자료(흐름)(MIMD)기계이다. 그러나 매개 주기에서 모든 처리기들이 같은 명령을 수행해야 한다면 오직 한명령흐름이 있는 단일명령(흐름), 다중자료(흐름)(SIMD)기계로 된다.

MIMD의 특수한 경우는 단일프로그람다중자료(SDMD)계산이며[363] 모든 처리기는 같은 프로그람을 실행하며 그 프로그람은 프로쎄스색인에 따르는 파라메터를 가진다. SIMD와 SPMD와의 차이는 SPMD계산에서 다른 명령들이 같은 주기에서 실행될수 있다는것이다.

동기성 이 속성은 처리기들이 얼마나 밀접하게 동기화되는가를 말해 준다. PRAM모형의 중심적인 특징은 동기적이라는것이다. 매 주기에서 *n*개 명령에 의한 모든 기억기 읽기동작은 어떤 처리기가 기억기쓰기 혹은 분기하기전에 수행되여야 한다.

PRAM은 명령준위에서 동기적이라고 말할수 있다. 이러한 강한 동기성은 SIMD기계를 제외하고 대부분 실제병렬콤퓨터들에는 없다. 오히려 실제적인 MIMD병렬콤퓨터는 비동기적이다. 매개 프로쎄스는 그자신의 단계로 실행되며 다른 프로쎄스의 속도와 무관계하다. 한 프로쎄스가 다른 프로쎄스로 하여금 정확한 의미를 가지도록 기다리게 한다면 보충적인 동기화조작이 수행되여야 한다.

이 두 극단사이에는 또 다른 동시도식이 있다. 흥미 있는것은 Valiant가 자기의 다량동기병렬(BSP)모형[623]을 제안한것이다. 매개 명령에서의 동기화대신에 매 초걸음들에서만 서로 동기화하는데 여기서 하나의 초걸음은 하나의 대량적인 명령으로서 고찰된다. 한 초걸음안에서 프로쎄스는 그외 명령들을 비동기적으로 실행한다. 또한 초걸음에서 모든 처리기로부터 수행되는 기억읽기조작은 어떤 처리기가 기억쓰기조작을 하기전에 수행되여야 한다.

또 하나의 성긴 동기화도식[253, 254]은 BSP모형과 대단히 류사하다. 병렬프로그람은 단계의 렬로서 나누어 진다. 한 단계는 단일배정명령문만큼 간단하거나 임의의 명령문의 렬만큼 복잡하다. 다중처리기는 매개 구안에서 비동기적으로 실행된다. 그것들은 매개 구의 끝에서 동기화된다.

호상작용기구 이 속성은 병렬처리기가 서로 동작에 영향을 주기 위해 어떻게 호상작용하는가를 보여 준다. PRAM모형에서 프로쎄스는 공유변수(또는 공유기억기)를 통하여 호상작용한다. 또 하나의 중요한 호상작용기구는 통보교환이다. 이 도식에서 프로쎄스들은 공유변수를 가지지 않는다(바꾸어 말하면 한 프로쎄스에 의하여 접근할수 있는 모든 변수들은 다른 프로쎄스에서 볼수 없다.).

대신에 프로쎄스들은 통보를 보내고 받는것에 의하여 호상작용한다. 공유변수를 통하여 호상작용하는 비동기 MIMD기계를 자주 **다중처리기**라고 부른다. 통보교환을 가지는

MIMD기계를 주로 **다중콤퓨터**라고 부른다. 이 호상작용기구는 여러가지 방법에 따라 결합될수 있다. 다중처리기와 다중콤퓨터는 5.4.1과 8.1.1에서 더 자세히 고찰한다.

주소공간 프로쎄스의 주소공간은 프로쎄스에 의하여 접근할수 있는 기억위치의 모임이다. 여러 기계(실례로 PRAM모형)들에서 모든 기억기위치는 프로그람작성자의 견지에서 보면 단일한 주소공간에 상주한다.

모든 병렬콤퓨터들은 단일주소공간을 가지지 않는다. 실례로 하나의 다중콤퓨터에서 매 처리기는 그자신의 개별적인 주소공간을 가진다. 이러한 기계를 **다중주소공간**을 가진다고 말한다. 이로부터 다중콤퓨터의 처리기들은 공유변수들을 통하는것이 아니라 통보교환에 의하여 통신한다.

리론적인 PRAM모형의 하나의 중요한 특징은 모든 처리기가 모든 기억위치에 대하여 동일한 접근시간을 가진다는것이다. 이러한 기계는 동일주기억기접근(UMA)을 가진다고 말한다.다른 한편 각이한 프로쎄스들이 단어위치에 접근하는데 각이한 량의 시간이 걸리면 기계는 비동일기억기접근(NUMA)을 가진다고 말한다. 이 기억모형은 5장과 8장에서 구체적으로 학습할것이다.

실례로 분산기억기공유(DSM)다중처리기에서 론리적인 공유기억기는 물리적으로 모든 처리기들에 분산되는데 이것을 **국부기억기**라고 부른다. 한 처리기의 국부기억기는 다른것의 원격기억기일수 있다. 어떤 콤퓨터들은 계층적으로 조직화된 기억기를 가진다. 국부기억기외에 처리기들은 대역적인 기억기에 접근한다. 하나의 국부단어에 접근하는 시간은 보통 원격 또는 대역기억기에 접근하는것보다 적으며 때때로 1~2차수이다. 따라서 NUMA기계들에서 자료와 프로그람에 대하여 다같이 국부성을 도입하는것이 중요하다.

기억기모형 이 속성은 기계모형이 공유기억기접근충돌을 어떻게 조종하는가를 규정한다. 몇가지 일치성규칙이 이런 충돌을 해소하는데 리용된다. PRAM모형에서 배타적읽기, 배타적쓰기(EREW)규칙을 사용할수 있는데 이 규칙에 따라 기억세포는 주기마다 최대로 한프로쎄스씩 읽을수 있고 쓸수도 있다. 병행읽기배타적쓰기(CREW)규칙은 매 주기에서 기억기세포를 다중처리기에 의해서 읽을수 있지만 쓰기는 기껏 하나의 처리기로 할수 있다. 병행읽기병행쓰기(CRCW)규칙은 매 주기에서 다중처리기가 같은 기억위치에 읽거나 쓰기를 둘 다 할수 있게 한다. 충돌하는 경우에 병행쓰기는 미리 규정된 접근방책에 따라 해소된다. 실제적인 다중처리기에서 공유기억기는 성능을 높이게 하는 기술들을 포함한 복잡한 체계이다. 따라서 병렬콤퓨터에서 기억기접근순서는 다중처리기가 지원하는것과 직접적으로 부합되지 않을수 있다. 공유기억기의 트랜잭션특성은 여러가지 기억기일치성규칙에 따라 설명되는데 보통 기억일관성모형으로 알려 져 있다. 이것은 5장에서 취급한다.

정의 1. 2. 원자동작이란 다음과 같은것이다.

- **비분할** 그것이 일단 시작되면 중간에 중단될수 없다는것이다. 이것은 다른 프로

쎄스의 중간상태를 볼수 없다는것을 의미한다.

- **유한** 그것이 일단 시작되면 유한시간크기내에 끝난다.

정의 1. 3. 원자동작의 더 강한 정의는 다음의 네가지 성질을 만족하여야 한다. 이러한 원자동작을 **트랜잭션**이라고 부른다.

원자성 트랜잭션은 정의 1.2를 만족시킨다. 그것을 다른 방식으로 보면 트랜잭션의 부분조작의 전체가 수행되거나 어떠한 부분조작도 수행되지 않는다.

일치성 트랜잭션은 언제나 무모순인 한 상태로부터 다른 상태에로 프로그람을 이동한다. 트랜잭션은 프로그람의미론에 의존된다. 실례로 단어쓰기조작은 절반단어를 쓰지 말아야 한다.

고립성 트랜잭션의 효과(결과)는 그것이 다른데로 넘어 갈 때까지 다른 트랜잭션에서 로출되지 말아야 한다.

견고성 다른데로 넘어 간다면 트랜잭션의 효과는 체계가 실패할 때까지 남아 있다.

1. 3. 2. 성능속성

의미속성은 프로그람작성자가 정확한 병렬프로그람을 쓰기 위하여 의미들을 리해해야 한다는 점에서 중요한 속성이다. 이 속성들은 상대적으로 오래동안 지속하며 보통 병렬콤퓨터의 한 세대로부터 다음세대에 이르기까지 같은것을 유지한다. 병렬콤퓨터에는 다른 속성들이 있는데 그것은 프로그람작성자가 효과적인 병렬프로그람을 개발하려면 알아야 한다. 이런 성능속성들은 고도로 가동환경에 관계되는것이며 보통 매 세대에서 개

표 1-4 **병렬체계성능속성**

용어	표기	단위
기계크기	n	차원 없음
박자속도	f	MHz
작업부하	W	Mflop
순차실행시간	T_1	S
병렬실행시간	T_n	S
속도	$P_n = W / T_n$	Mflop/s
속도증가	$S_n = T_1 / T_n$	차원 없음
효률성	$E_n = S_n / n$	차원 없음
리용	$U_n = P_n / (nP_{peak})$	차원 없음
시동시간	t_0	μs
접근대역너비	r_∞	MB/s

선된다. 기본성능속성들이 표 1-4에 제시되였다. 기계크기 n은 한 병렬콤퓨터의 처리기수이다. 작업부하 w는 한 프로그람에서 계산조작의 수이다. P_{peak}는 처리기의 최대속도이다.

정의 1. 4 병렬프로그람에는 세가지 형태동작이 있다. 계산동작은 산수/론리, 자료전송, 순차형콤퓨터에서 볼수 있는 조종흐름동작을 포함한다.

병렬성동작은 생성과 정지, 문맥절환, 그룹화와 같은 프로쎄스를 관리하는데 필요되며 호상작용동작을 통신하고 프로쎄스들을 동기화하는데 요구된다.

병렬성과 호상작용동작은 명시적이거나 암시적일수 있다. 명시적동작은 병렬프로그람본문에서 나타나는것이다. 실례로 프로쎄스는 UNIY함수 fork( )를 호출하여 명시적으로 생성될수 있다. 암시적동작은 프로그람에서 나타나지 않지만 체계에 의하여 암암리에 수행된다.

실례로 프로쎄스의 시간토막이 실행에서 벗어 나면 조작체계는 프로쎄스교환이 현재의 프로쎄스를 대기렬안에 놓고 다른 프로쎄스가 실행되게 한다. 이러한 프로쎄스교환은 병렬프로그람에서 명시적으로 규정되지 않는다. 명시적 그리고 암시적인 동작모두는 실행하는데 시간이 걸린다.

정의 1. 5 병렬성과 호상작용동작은 순수 계산작업부하를 실행하는데 요구되는 시간외에 수행하는데 추가시간이 요구된다는 점에서 부가처리의 원인으로 된다. 특히 이 부가처리들은 다음의 네 단계로 나눌수 있다.

- 프로쎄스관리에 의하여 일어 나는 병렬성부가처리
- 처리기들이 정보를 교환할 때의 부가처리
- 동기화동작을 수행하는데서의 동기화부가처리
- 어떤 프로쎄스가 다른 프로쎄스들이 동작하는 동안 기다릴 때에 초래되는 부하불균형부가처리

부가처리가 없어 n처리기를 사용할 때 속도증가는 언제나 n이다.

이러한 부가처리는 3장에서 자세히 취급될것이다. 지금은 한 마디로부터 다른 마디로 한개의 통보를 통신하기 위한 두개의 부가처리척도를 간단히 정의한다(처음으로 Hockney가 도입하였다[313].).

이러한 통신을 **점대점통신**이라고 부른다. 집중통신이라고 부르는 다른 형태의 통신이 있는데 하나이상의 통보문이 다중마디에서 동시에 통신된다. 시동시간은 t_0으로 표시하며 하나의 0byte 또는 짧은 통보문(실례로 한 단어)을 통신하는 시간이다. 1μs단위로 잰다. 시동시간은 역시 **통신지연시간**이라고 부르며 또는 간단히 **지연시간**이라고 한다. 점근대역너비는 $r\infty$로 표시하는데 긴 통보문을 통신하는 비률(MB/S로 잰다.)이다. 이 용어들은 점대점 그리고 집중통보문넘기기동작들에서 전체 통신시간을 평가하는데 사용된다.

1. 3. 3. 추상기계모형

추상기계모형은 주로 물리기계의 세부가 필요 없는 병렬알고리듬의 설계와 해석에서 쓰인다.

아래에서 세개의 추상모형을 본다. 매 모형은 서로 다른 가정하에서 많은 변종을 가진다.

PRAM모형 이미 1.3.1에서 PRAM모형의 의미속성을 서술하였다. 그것의 성능속성을 보면 다음과 같다.

- 기계크기 n은 임의의 크기일수 있다.
- 기본시간단계를 **주기**라고 부른다.
- 한 주기내에서 매 처리기는 반드시 하나의 명령을 수행한다. 명령은 빈 명령일수 있는데(다시 말하면 아무것도 하지 않는다.) 이 경우에 처리기는 그 주기에서 휴식한다고 말한다.
- 모든 처리기들은 매 주기에서 암시적으로 동기화되며 동기화부가처리는 령이라고 가정한다. 통신은 공유변수의 읽기쓰기를 통하여 진행된다. 통신부가처리는 무시된다. 병렬성부가처리도 무시된다. 따라서 PRAM프로그람에서 유일하게 고려되는 부가처리는 작업불균형부가처리이다.
- 명령은 임의의 우연접근명령일수 있다[16]. 실례로 명령은 한주기에서 다음의 세가지 동작을 수행한다.
  - 기억기로부터 연산수 하나 또는 두 단어를 꺼내며
  - 산수론리연산을 수행하고
  - 결과를 기억기에 저장한다.

실례 1.4. PRAM기계우에서 병렬실행

2개의 N차원벡토르 A와 B의 스칼라적 S를 EREW PRAM콤퓨터에서 계산하기 위하여서는 $2N/n$주기에서 국부결과를 얻기 위해 매 처리기에 $2N/n$개의 더하기와 곱하기를 배정할수 있다. 나무형감소법을 써서 $\log n$주기내에 n개의 국부합을 최종합 S에 더할수 있다. 전체 실행시간은 $2N/n+\log n$주기이다. $2N$개 주기를 가지는 순차알고리듬과 비교하면 병렬PRAM알고리듬들은 속도증가가 $N \gg n$일 때 $n/\{1+[n/(2N)]\log n\} \rightarrow n$으로 된다.

순차알고리듬의 시간복잡성은 한파라메터 즉 문제크기 N의 함수이다. 병렬 PRAM알고리듬은 또 다른 파라메터 즉 기계크기 n을 도입한다. 대부분의 PRAM알고리듬의 시간복잡성은 N과 n의 함수로 표현된다. PRAM모형은 작업불균형부가처리를 고려한다. 실례로 우의 병렬성알고리듬의 마지막단계에서 오직 한 처리기만이 일하면서 최종합을 계산하고 이때 다른 처리기는 대기상태에 있다.

그것은 간단하고 의미가 명백하므로 PRAM모형은 지금까지 많은 콤퓨터과학자들에게 익숙해 졌다. 리론적병렬알고리듬의 대부분은 PRAM모형 또는 그것의 변종에 맞게 규정된다. 이 모형은 역시 병렬알고리듬의 복잡성을 해석하는데 널리 쓰인다.

PRAM모형의 주되는 결함은 령통신부가처리와 명령준위동기에 대한 비현실적인 가정에 있다.

PRAM모형과 관련한 또 하나의 문제는 PRAM알고리듬의 시간적복잡성이 자주 큰 0표기법으로 표현된다는것이다. 이것은 자주 혼돈되는데 그것은 기계크기 n이 보통 현재 병렬콤퓨터들에서 작기때문이다. 실제적인 기계성능을 평가하는데서 큰 복잡성에 절대로 의거하지 말아야 한다.

실례 1.5. PRAM단계들에서 계산복잡성

세개의 PRAM알고리듬 *A, B, C*가 시간복잡성 $7n$, $(n\log n)/4\ n\log n$을 가지고 n−처리기 PRAM콤퓨터우에서 실행된다고 하자. big표기법에 따르면 알고리듬 *A*가 가장 빠르고(n) 그다음 $C((n\log\log n))$이며 제일 느린것은 $B((n\log n))$이다.

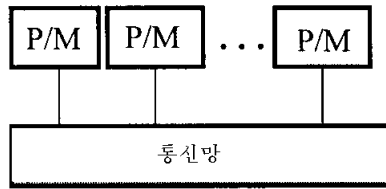
사실상 1024개의 처리기보다 크지 않은 처리기를 가지는 기계들에서 $\log n \le \log 1024=10$이고 $\log\log n \le \log\log 1024<4$이다. 따라서 1024개보다 작은 처리기를 가진 체계들에서 가장 빠른 알고리듬은 실지로 *B*이고 그다음은 *C*이다. 그리고 가장 느린것은 *A*이다. PRAM모형에 기초한 병렬정렬알고리듬은 $O(n)$항목들을 $O(\log n)$시간내에 정렬할수 있다. 그러나 실제적인 병렬콤퓨터에서 정렬화는 이러한 "최량화"된 알고리듬을 쓰지 않는다. 왜냐하면 큰 표기법은 큰 상수인자를 숨기기때문이다(반영하지 않는다.). 게다가 이 $O(\log n)$알고리듬이 개선될 때조차 여전히 실제적인 사용에 적합치 않는데 그것은 모든 부가처리를 무시하기때문이다.

PRAM모형은 정확치 않음에도 불구하고 많은 세부적인것들을 추상화하며 높은 수준의 병렬알고리듬을 개발하는데서 우월한 모형으로 된다. 많은 병렬알고리듬들은 PRAM모형을 써서 개발되므로 실제적으로 훌륭한 알고리듬으로 된다. 때때로 비현실적인 PRAM알고리듬은 실제적인 알고리듬으로 개선될수 있다. 그것들은 모두 부가처리를 수행하기 위한 보충적파라메터들을 도입하며 PRAM모형보다 좀 더 복잡하다. 모든 이 추상모형들에 공통적인 특징은 그것들이 통신망위상에 무관계하다는것이다.

BSP모형 다량동기병렬모형(BSP)은 하바드종합대학의 Leslie valiant에 의하여 PRAM모형의 결함을 극복하기 위해서 창안되였다. 한편 이것은 PRAM의 성질을 유지하고 있다. BSP콤퓨터는 그림 1−4에서 보는바와 같이 통신망에 의해서 호상접속되는 n개처리기/기억쌍(마디)들의 모임으로 구성된다.

BSP프로그람은 n프로쎄스를 가지며 매개 프로쎄스는 하나의 마디우에 상주한다. 기본시간단위는 주기(또는 시간단계)이다. 프로그람은 초걸음의 엄밀한 렬로서 실행된다.

매개 초걸음안에서 프로쎄스는 최대로 n주기내에 계산동작을 수행하며 통신동작은 gh주기에 걸쳐 실행되고 하나의 장벽(barrier)동기화는 l주기에 걸쳐 수행된다. 장벽은 모든 프로쎄스들이 그것의 어느 하나가 다음초걸음을 시작하기전에 현재 초걸음을 끝내도록 기다리게 한다.

BSP콤퓨터는 MIMD체계이다. 왜냐하면 프로쎄스들은 각이한 명령들을 동시에 수행할수 있기때문이다. 그것이 초걸음준위에서 완만하게 동기화되는데 비해서 명령준위는 PRAM모형에서 밀접하게 동기화된다. 한초걸음내에서 각이한 프로쎄스들은 그자체의 단계에서

w: 매 초고속단계에서 최대계산시간

l: 장벽동기화부가처리

g: *h* 관계결수

- MIMD
- 초고속단계 :
  계 산
  통 신
  장 벽
- 변수grain
- 성긴 동기
- 비령부가처리
- 통보문통과 혹은 공유변수

그림 1-4. 다량동기병렬모형

비동기적으로 실행된다. BSP모형은 어떤 규정된 호상작용기구를 요구하지 않는다. 즉 그것은 공유변수이든지 통보문넘기기일수 있다. 하나의 주소화공간이 있으며 하나의 처리기는 그것의 국부기억뿐아니라 다른 마디의 어떤 원격기에도 접근할수 있다. 한초걸음내에서 매개 계산조작은 오직 그 국부기억안의 자료만을 사용한다. 이 자료들은 프로그람 출발시에 또는 이전 초걸음의 통신조작에 의하여 국부기억안에 놓인다.

그렇기때문에 한프로쎄스의 계산동작은 다른 프로쎄스와 무관계하다. 통신은 언제나 점대점방식으로 실현된다. 따라서 다중프로쎄스에 대하여 같은 주기안에서 같은 기억위치를 읽거나 쓰는것은 허용되지 않는다.

장벽동기화로 하여 한 초걸음안의 모든 기억 및 통신동작은 다른 초걸음의 임의의 동작을 시작하기전에 완전히 끝나야 한다. 이 모든것은 BSP콤퓨터가 순차적인 무모순기억기모형을 가진다는것을 의미한다.

실제적인 병렬콤퓨터가운데서 각이한 통신형태를 요구하는 요소통신동작들이 있다. 간단히 하기 위하여 BSP모형은 h관계개념에 의하여 하나의 BSP초걸음안에서 통신동작들을 추상화한다. 이것은 성능예측과 알고리듬복잡성해석을 간단히 한다.

정의 1. 6 h관계는 어떤 통신동작의 추상화로서 매개 마디는 여러 마디들에 최대로 h단어들을 보내며 매개 마디는 최대로 h단어들을 받는다. BSP콤퓨터에서 h관계를 실현하는 시간은 gh주기보다 크지 않으며 여기서 g는 기계가동환경에 따라 결정되는 상수이다.

BSP모형은 프로쎄스관리에서 병렬성부가처리를 제외하고 모든 부가처리를 고려하므로 PRAM모형보다 더 현실적이다. 한 초걸음의 실행시간은 다음과 같이 결정된다.

- 부하불균형을 고려할 때 계산시간 w는 어떤 처리기가 계산동작에 소비한 주기의 최대수이다.
- 동기화부가처리는 l이며 그것은 통신망대기시간의 아래한계를 가지므로(바꾸어 말하면 물리적인 망을 통하여 한 단어를 넘기기하기 위한 시간) 언제나 령보다 크다.
- 통신부가처리는 gh주기이며 여기서 g는 h관계를 실현하는 비례결수이다. G의 값은 가동환경에 관계되지만 통신형태와 무관계하다. 다른 말로 말해서 gh는 시

간을 소비하는 h관계를 실행하는 시간이다. 파라메터 g는 보다 효과적인 통신 지원을 가지는 콤퓨터에서 작은 값을 가진다.

- 초결음시간은 합 $w+gh+l$에 의하여 평가된다.

BSP모형은 한 초결음안에서 계산, 통신, 동기화동작들의 중첩을 허락하지 않는다. 동작의 세가지 형태는 모두 충분히 중첩되며 한 초결음시간은 max(w, gh, l)로 된다.

그러나 보다 긴 $w+gh+l$만을 리용한다.

실례 1.6. BSP기계모형을 사용하는 병렬실행

실례 1.4에 있는 스칼라적문제를 고찰하자. 이 문제를 8개의 처리기 BSP콤퓨터우에서 4개의 초결음내에 풀수 있다.

초결음 1

계산 : 매 처리기는 $w=2N/8$ 주기내에 그것의 국부합을 계산.

통신 : 처리기 0, 2, 4, 6은 국부합을 처리기 1, 3, 5, 7에 보내고 관계 1을 적용

장벽동기화.

초결음 2

계산 : 처리기 1, 3, 5, 7 매개는 한개의 더하기 ($w=1$)를 수행

통신 : 처리기 1, 5는 중간결과를 처리기 3과 7에 보내고 관계 1을 적용

장벽동기화.

초결음 3

계산 : 처리기 3, 7은 한개의 더하기 ($w=1$)를 수행

통신 : 처리기 3은 중간결과를 처리기 7에 보내고 관계 1을 적용

장벽동기화.

초결음 4

계산 :처리기 7은 최종합을 생성하기 위해 한개의 더하기($w=1$)를 수행

통신이나 동기화는 요구되지 않는다.

전체 실행시간은 2N/8+3g+3l+3주기이다. 일반적으로 n-처리기 BSP우에서 실행시간은 2N/n+logn(g+l+1)주기이다. 이것은 PRAM콤퓨터우에서 2N/n+lgn시간 걸리는것과 대조적이다. 두개의 나머지항 glogn과 $l$logn은 각각 통신 및 동기화부가처리에 대응한다.

위상병렬모형 저자들은 위상병렬모형[657]을 병렬계산을 위하여 제안하였는바 그것은 더우기 우의 두 추상기계모형을 세련시켰다.

이 모형은 BSP모형과 류사하며 다음 차이를 가진다.

- 병렬프로그람은 위상의 렬로 실행된다. 다음단계는 현재단계에 있는 모든 동작들이 끝날 때까지 시작할수 없다. 위상에는 세가지 형태가 있는데 아래와 같이 정의된다.

  병렬성위상 : 이것은 부가처리작업이 처리관리에서 동반된다는것을 가리키는데 그것은 프로쎄스생성 및 병렬처리를 위한 그룹화이다.

  계산위상 : 하나 또는 그이상의 처리기는 일정한 수의 국부계산동작을 수행한다. 여기서 국부라는것은 마디에 필요한 모든 자료들이 그것의 국부기억안에 있다는것을 의미한다.

  호상작용위상 : 이것은 과제들의 호상작용동작을 수행하는데 필요된다는것을 가리키며 그 동작은 동기화 또는 집합(실례로 감소 또는 주사)조작, 통신조작이다.

- 주어 진 계산위상에서 w와 t_f에 의하여 작업부하를 표시하며(실례로 Mflop로) 그것은 계산동작을 실행하는 평균시간(μs)이다. 서로 다른 계산위상은 각이한 속도로 각이한 작업부하를 실행하는데 그것들은 서로 다른 w와 t_f값에 대응된다. 이것은 이전 두 모형과 대비되는데 여기서 매개 동작은 하나의 동일한 주기를 취한다고 가정하였다.
- 각이한 호상작용동작은 각이한 시간이 걸린다. 그러나 호상작용동작을 수행하는 시간에 대한 일반적인 형식이 있다. 즉

$$T_{interact}(m,n) = t_0(n) + \frac{m}{r_\infty(n)} = t_0(n) + m \cdot t_c(n) \tag{1.1}$$

여기서 m은 통보문의 길이라고 부르는데 byte로 잰다. 시동시간 $t_0(n)$과 비동기대역 $r_\infty(n)$은 기계크기 n의 함수이다. 파라메터 $t_c(n) = 1/r_\infty(n)$을 byte당 **통보문시간**이라고 부른다. 아래에서 병렬성, 계산, 호상작용위상의 순차에 대하여 설명한다. BSP모형과 류사하게 렬을 **초결음**이라고 부른다. 호상작용이 mB의 통보문길이를 가지고 통신하는 경우에 초점을 둔다. 전부는 아니지만 어떤 위상들은 한 초결음안에서 빠질수 있다. 실례로 초결음은 바로 하나의 계산위상을 포함하며 한편 다른 초결음은 꼭 하나의 호상작용위상를 포함할수 있다.

실례 1.7. 위상병렬모형을 리용하는 병렬실행

실례 1.4의 스칼라적문제를 다음의 3-위상병렬알고리듬으로 푼다.

첫번째 위상은 n프로쎄스를 생성한다. 두번째 위상에서 프로쎄스 i는 모든 $0 \le i \le n$에 대하여 eotjdy LocalSum[i]를 계산한다. 세번째 위상에서 모든 프로쎄스들이 최종결과 S를 생성하는 감소동작에 참가한다.

계산위상에서 매개 프로쎄스는 $1/t_f$의 속도로 $2N/n$ flop의 작업부하를 계산하며 여기서 t_f는 한 처리기가 한 flop를 수행하는 평균시간이며 단일처리기속도의 거꿀수와 같다.

즉 t_f=1/p이다.

```
병렬위상   :   parfor (i=0; i<n; i++)
                 {
계산위상    :      (localSum)[i] = Σ_{j=in}^{(i+1)n} A[j] • B[j]
호상작용위상:      s = sum_reduction (LocalSum[i]);
                 }
```

그림 1-5. 위상병렬모형을 리용한 병렬스칼라적계산

병렬성작업부하는 $t_p(n)$이고 감소동작을 위한 호상작용작업부하는 $t_0(n)$이라고 가정하자(감소동작은 상수통보문길이 m=4B을 가진다.). 그러면 전체 실행시간은 $2Ntf/n+tp(n)+t_0(n)$이다.

초결음에서 n개 처리기에 의하여 수행되는 전체 계산작업부하가 $W=nw$라고 가정하자. 그것은 n개 독립계산들로 나눌수 있다. n개의 독립계산의 실행시간은 우연변수로 가정되며 그것은 wtf의 평균값을 가지며 표준편차는 δt_f이다. 그렇기때문에 파라메터 w와 δ는 각각 평균값과 표준편차를 표현한다. 직관적으로 w는 립도이며 δ는 작업불균형을 각각 가리킨다. 작은 δ는 근사적으로 균형화된 병렬계산위상을 가리킨다. δ=0일 때 전체 작업부하 w는 처리기들안에서 골고루 분할된다.

n개 처리기들이 서로 무관계한 계산들을 수행하므로 계산단계의 실행시간은 n개의 처리기의 개별적인 실행시간들가운데서 최대값으로 되여야 하며 이것은 T_{comp}로 표시된다. Kouskal과 Weiss[379]의 최대값이

$$T_{comp} = \left(w + \sigma\sqrt{2\log n}\right) t_f \qquad (1.2)$$

에 의하여 근사된다.

정의 1.7 초결음의 호상작용위상에서 통보문길이가 $m=\alpha wB$라고 가정하자.

파라메터 α를 초결음의 통신 대 계산비률(CCR)이라고 부르며 flop계산당 통신되는데 몇 byte가 필요한가를 측정한다.

식 (1.1)로부터 호상작용작업부하는

$$T_{interact} = t_0(n) + m/r_\infty(n) = t_0(n) + \alpha \cdot w \cdot t_c(n) \qquad (1.3)$$

으로 표현된다.

n처리기우에서 초결음의 전체 실행시간은

$$T_n = T_{comp} + T_{interact} + T_{par} = \left(1+\sigma\sqrt{2\log n}\right)t_f + t_0(n) + \alpha \cdot \omega \cdot t_c(n) + t_p(n) \quad (1.4)$$

에 의하여 표현된다.

위상병렬모형은 PRAM과 BSP모형을 개선하였으므로 실제적인 기계/프로그람트랜잭션을 포괄하는데 더 편리하다. 모든 작업부가처리형태들은 식 (1.4)에서와 같이 계산된다. 즉 부하불균형부가처리(δ항), 호상작용부가처리(t_0과 t_c항), 병렬성부가처리(t_p항)들이 계산된다. 각이한 호상작용에는 각이한 식들이 사용된다.

표 1-5 위상병렬모형의 파라메터

파라메터	단위	의미	의존성
t_0	μs	시동시간	체 계
t_c	μs/B	1byte를 통신하는 시간	체 계
t_f	μs	1Flop 실행하는 시간	체계와 응용
t_p	μs	병렬부가처리	체 계
w	Mflop	평균작업부하	응용
σ	Mflop	작업부하의 표준편차	응용
α	B/flop	통신 대 계산비률	응용

이 식들은 호상작용조작들을 실제 기계우에서 측정하여 얻을수 있다. 일단 얻으면 그것들은 각이한 응용들에서 여러번 사용할수 있다. 이러한 내용은 3장에서 다시 취급된다. 위상병렬모형에서 리용된 파라메터들은 표 1-5에서 개괄된다.

1. 3. 4. 물리기계모형

대규모콤퓨터체계들은 일반적으로 여섯개의 실천적인 기계모형으로 분류되는데 그것들은 다음과 같다. 단일명령다중자료(SIMD)기계, 병렬벡토르처리기(PVP), 대칭다중처리기(SMP), 대용량병렬처리기(MPP), 워크스테이션클라스터(COW), 분산공유기억기(DSM)다중처리기들이다. SIMD콤퓨터들은 대부분 특수목적의 응용들에서 사용된다. 기타 모형들은 모두 MIMD기계들이며 표 1-6에서 제시한다.

다섯개의 병렬모형을 그림 1-6에서 보여 주었다. 대부분의 현대 병렬콤퓨터들은 상업적으로 쓸수 있고 즉시에 구입할수도 있다. 그것은 하드웨어와 쏘프트웨어구성요소들로 만들어 졌다. 오직 례외로 되는것은 PVD기계이며 많은 제작블로크들은 전용으로 만들어 졌다.

표 1-6 병렬기계모형의 의미속성

속성	PRAM	PVP/SMP	DSM	MPP/COW
동차성	MIMD	MIMD	MIMD	MIMD
동기화	명령준위동기	비동기 혹은 성긴 동기	비동기 혹은 성긴 동기	비동기 혹은 성긴 동기
호상작용기구	공유변수	공유변수	공유변수	통보문넘기기
주소공간	단일	단일	단일	다중
접근비용	UMA	UMA	NUMA	NORMA
기억기모형	EREW, CREW or CRCW	순차일치성	약한 순서화가 널리 쓰인다	N/A
기계실례	리론적모형	IBM R50, Cray T-90	Stanford DASH, SGI Origin 2000	Cray T3E, Berkeley NOW

병렬벡토르처리기 전형적인 병렬벡토르처리기들인 PVP의 구조를 그림 1-6 ㄱ)에서 보여 준다. PVP의 실례들로서 Cray C-90, Cray T-90, NEC SX-4가 있다. 이 체계들은 적은 수의 주문설계된 벡토르처리기(VP)들을 포함하며 매개는 최소한 1Gflop/s성능을 가진다.

주문설계된 고대역크로스바교환기는 이 벡토르처리기들을 얼마간의 공유기억기(SM)모듈들에 련결하는데 그것은 고속자료접근을 제공한다. 실례로 T-90에서 공유기억기는 자료를 하나의 처리기에 14GB/s로 넘길수 있다. 이러한 기계는 보통 캐쉬를 사용하지 않으며 오히려 많은 벡토르등록기와 명령완충기를 사용한다.

대칭다중처리기 SMP의 구조를 그림 1-6 ㄴ)에서 보여 주었다. 그 실례로 IBM R51, SGI Power Challenge, DEC Alpha Server 8400이 있다. PVP와 달리 SMD체계는 소편내장캐쉬와 소편외장캐쉬를 가지는 극소형처리기들을 사용한다. 이 처리기들과 공유기억에 고속 snoopy모선을 통하여 련결된다. 어떤 SMP들에서는 크로스바교환기의 모선에 보충적으로 사용된다.

SMP체계들은 상업적응용들에서 많이 사용되는데 그 응용들은 자료기지,닫긴 트랜잭션체계, 자료보관고들에서 리용된다. 체계가 대칭이라는것이 중요하며 거기서 매개 처리기는 공유기억, I/O장치들과 조작체계봉사에 대하여 동일한 접근을 가진다. 대칭이면 고수준의 병렬성을 실현할수 있다. 그것은 비대칭(또는 주종관계)다중처리기체계에서는 불가능하다. 1997년에 대부분PVP와 SMP들은 최대로 64처리기를 가지고 있었고 그것은 Sun Uitra Enterprise 1000에서 볼수 있다. 한계는 주로 중심화된 공유기억과 모선 또는 직교체계와 호상련결되여 쓰이는데 기인되며 그것은 한번 만들어 졌을 때 둘 다 확대하기가 불가능하다.

대용량병렬처리기 과학계산, 공학모의, 신호처리, 자료창고관리 등과 같은 응용들에서 리용된다. 고도의 병렬성의 우점을 살리려면 분산기억구성방식을 개발하여 보다 높은

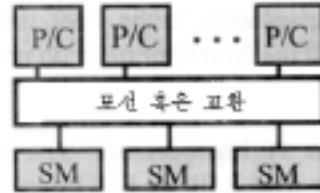
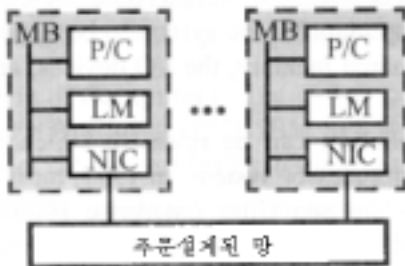
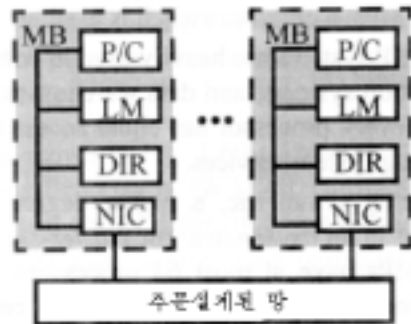
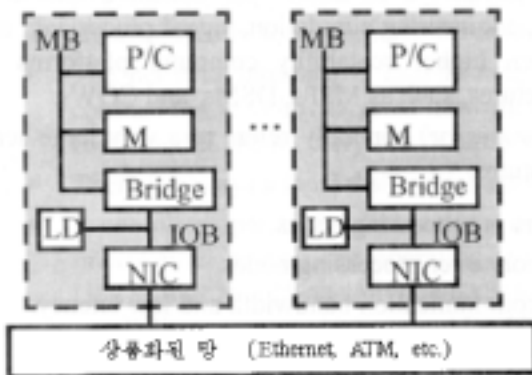



그림 1-6. 다섯개의 물리적병렬콤퓨터모형 : PVP, SMP,MPP,DSM와 COW

확대가능성을 리용하여야 한다. 그 구조들은 MPP와 DSM 그리고 COW들이다.

MPP(대용량병렬처리기)는 다음의 특징을 가지는 대단히 큰 규모의 콤퓨터체계를 가리킨다.

- 프로쎄스마디들에서 상품극소형처리기들을 리용한다.
- 프로쎄스마디들전체에 대하여 물리적으로 분산된 기억을 리용한다.
- 높은 통신대역너비와 낮은 지연시간을 가진 호상접속을 리용한다.
- 수백 또는 지어 수천개의 처리기들로 확대할수 있다.
- 다중처리모형과 마찬가지로 MPP는 비동기MIMD이다. 그러나 프로쎄스는 공유

변수동기동작이 아닌 블로크통보문과 동작을 통하여 동기화된다.

- 프로그람은 다중프로쎄스들로 구성되며 매개는 전용주소공간을 가진다. 프로쎄스들은 통보문을 통과하여 호상작용한다.

그림 1-6 ㄷ)에서 모형화된 MPP는 보다 제한적인데 Intel Paragell과 TFLOP와 같은 기계를 표현한다. 이런 기계는 얼마간의 프로쎄스마디들로 이루어 지고 매개는 하나 또는 그이상의 극소형처리기를 포함하며 극소형처리기들은 고속기억기모선에 의하여 국부기억과 망대면부회로(NIC)에 호상접속된다. 그 마디들은 하나의 고속, 독점통신망에 호상련결된다. 이 마디들을 밀집결합된다고 말한다.

분산공유기억기(DSM)기계 DSM이 그림 1-6 ㄹ)에서 모형화되고 있는데 Stanford DASH구조[405]에 기초하고 있다. 캐쉬등록부(DIR)는 분산일관성캐쉬를 지원하는데 사용된다. Cray T3D는 역시 DSM기계이다[3]. 그것은 일관성캐쉬를 실현하는데 사용하지 않는다. 대신에 T3D는 특수한 하드웨어와 쏘프트웨어의 확장에 의거하여 임의의 블로크크기준위에서 DSM에로 확장되며 크기준위는 단어로부터 공유자료의 큰 페지에 이르기까지 포괄된다.

DSM기계와 SMP기계사이의 주요차이는 기억이 물리적으로 각이한 마디들가운데서 분산된다는것이다. 그러나 체계하드웨어와 쏘프트웨어는 응용사용자들에게 단일한 가상주소공간을 생성해 준다. DSM기계는 역시 TreadMarks와 같은 워크스테이션망우에서 쏘프트웨어확장으로 실현될수 있다. TreadMarks는 10장에서 고찰하게 된다.

콤퓨터의 클라스터 클라스터의 개념은 그림 1-6 ㅁ)에서 보여 준다. 실례로서 Digital TruCluster[197], IBMSP2[14], BerkeleyNOW[178]이다. 클라스터는 어떤 경우에는 저비용MPP의 변종이다. 클라스터특징에서 중요한것을 아래에 서술한다.

- COW의 매개 마디는 완전한 워크스테이션으로서 몇가지 주변기구(실례로 화면, 건반, 마우스 기타 등등)를 가지지 않는다. 이러한 마디를 때때로 "**머리 없는 워크스테이션이다**"라고 부른다. 마디는 역시 SMP PC로 된다.
- 마디들은 저비용상업망(즉 Ethernet, FDDI, Fiber-Channel, ATM 교환)을 통하여 련결된다. 하지만 독점적인 망이 몇가지 상업적클라스터들에서 사용된다.
- 망대면부는 마디안에서 I/O모선에 성긴 쌍결합된다. 이것은 MPP가(그림 1-6 ㄷ) 밀집결합된 망대면부를 가진다는것과 관련된다. 그리고 MPP의 망대면부는 프로쎄스마디의 기억모선에 련결된다.
- 항상 국부디스크를 가지며 그것은 MPP마디에서는 존재하지 않는다.
- 매개 마디에는 완전한 조작체계가 있으며 한편 어떤 MPP들에서는 오직 하나의 마이크로핵심부만이 존재한다. COW의 조작체계는 동일한 워크스테이션 UNIX에 단일체계상과 리용가능성, 병렬성, 통신, 부하균형을 지원하기 위해 부가쏘프트층을 더한것이다.

MPP와 COW사이의 경계는 요즈음 모호해 지고 있다. IBM SP2은 MPP로 고찰된다. 그러나 그것은 클라스터구조를 가지며 독점고성능교환은 통신망으로 리용된다. 클라스터화는 확대가능한 병렬콤퓨터들을 개발하는데서 하나의 추세로 되는데 그런 콤퓨터들은 다음부분에서 강조된다.

1. 4. 클라스터화의 기본개념

이 부분에서는 콤퓨터의 클라스터와 관련된 기본개념들에 대하여 더 구체적으로 고찰한다.

다중콤퓨터클라스터는 아래에서 SMP와 MPP분산형체계들과 갈라 볼수 있다. 우점과 문제점들은 클라스터를 만드는데서 찾아 볼수 있다. 세부적인 망기술은 6장에서 고찰된다. 클라스터화의 원리는 9장에서 고찰되며 클라스터체계의 실례는 10장에서 고찰된다.

1. 4. 1. 클라스터의 특성

클라스터는 완전한 콤퓨터(마디들)의 집합체이고 그 집합체는 물리적으로 고성능망 또는 국부망(LAM)에 의해서 호상련결된다. 전형적인 콤퓨터마디 SMP봉사기로는 워크스테이션개인용콤퓨터이다. 보다 중요하게 모든 클라스터마디들은 서로 하나의 집적화된 계산자원처럼 집합체로서 동작할수 있어야 하며 그 과정에 개별적으로 호상작용하는 리용자들이 매개 마디를 리용하는데서 사용자들의 요구를 충족해야 한다.

다섯개의 구조적개념들이 전체 콤퓨터를(마디) 하나의 호상련결된 모임으로 하여 하나의 클라스터안에 포함된다. 그 콤퓨터들은 련속적으로 효과적인 (성능)봉사를 제공하는 하나의 체계와 같이 집합체처럼 작용한다. 클라스터의 개념적구조를 그림 1-7에서 보여 주었다.

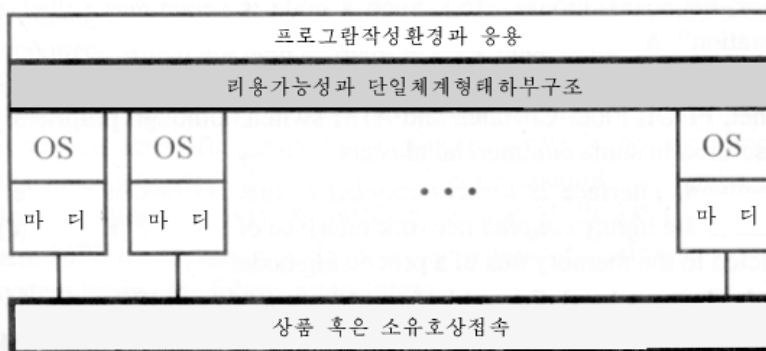



그림 1-7. 다중콤퓨터의 대표적인 클라스터구성방식

클라스터에 관한 다섯개의 구조적개념들을 아래에 서술한다. 나머지는 9장과 10장에서 고찰한다.

- **클라스터마디** 매개 마디는 완전한 콤퓨터이다. 이것은 매개 마디가 처리기들,

캐쉬들, 여러개의 I/O적응기들을 가진다는것을 의미한다. 게다가 완전한 표준조작체계는 매개 마디에 있다. 마디는 하나이상의 처리기를 가질수 있지만 오직 OS상에서는 하나의 복사를 가질수 있다. 10장에서는 몇가지 클라스터마디들의 일부를 서술할것이다.

- **단일체계영상** 클라스터는 단일계산자원이다. 이것은 분산체계와 대조적인데(실례로 국부콤퓨터망) 여기서 마디들은 오직 개별적인 자원으로서 사용된다. 클라스터는 몇가지 단일체계(SSI)기술에 의하여 실현된다.
  SSI는 클라스터를 사용하기 쉽고 관리하기 쉽게 한다. 지금까지 상업적으로 리용할수 있는 클라스터들은 충분한 SSI조작의 정도를 달성하지 못하였다.
- **마디간 련결** 클라스터의 마디들은 Ethernet FDDI, Fiber−Channel들이다. ATM교환과 같은 망을 통해서 련결된다. 그러나 표준규약들은 마디간 통신에 사용된다. 클라스터를 구성하기 위한 각이한 망들은 6장에서 고찰된다.
- **높아 진 유용성** 클라스터는 체계의 유용성을 높이는 비용효과적인 방도를 제공해 주는것으로써 체계의 유용성은 체계가 사용자에게 유용하게 되는 시간률이다. 유용성기술은 9장에서 고찰된다.
- **높은 성능** 클라스터는 일련의 봉사령역에서 높은 성능을 제공해야 한다. 그 하나의 령역은 초봉사기로서 클라스터를 취급하는것이다. n−마디클라스터의 매개 마디가 m개 의뢰기에게 봉사한다면 클라스터는 전체적으로 mn의뢰기에 동시에 봉사할수 있다.

다른 령역은 분산병렬처리에 의해 큰 단일일감을 실행하는 시간을 최소화하는데 클라스터를 리용하는것이다.

1. 4. 2. 구성방식의 비교

클라스터, SMP, MPP분산체계들은 네개의 중복된 구조로서 그림 1−8에서 그것을 보여 주었다. 분산체계의 실례는 LAN이다. LAN에서 마디는 PC, 워크스테이션 또는 SMP봉사기일수 있다. 마디복잡성은 하드웨어와 쏘프트웨어와 관련된다. 클라스터마디는 MPP마디보다 복잡하다. 왜냐하면 클라스터는 디스크와 독립적인 조작체계를 가지며 MPP는 디스크를 가지지 않을수 있고 또 오직 극소핵심부만을 사용하기때문이다.

SMP봉사기는 클라스터가 말단기, 인쇄기, 외부확장불가능 디스크여유묶음(RAID), 테프장치와 같은 많은 주변장치를 가지므로 대단히 복잡하다. 이 주변장치가운데서 어떤것들은 클라스터마디에 없을수도 있다. 앞으로 클라스터경계는 MPP와 SMP가 더 많이 중첩되게 련속적으로 확장될것이라고 보고 있다.

수평축은 여러가지 추상준위 즉 한 응용으로부터 부분체계, 실행체계, OS핵심부, 하드(기억과 I/O)준위까지 이르는 여러 추상준위에서 SSI의 정도를 보여 준다. 다시 말하여 SSI는 상대적개념으로서 사용자의 관점으로부터 SSI의 경계에 관계된다. SMP는 언제나 모든 준위에서 SSI를 제공한다. MPP는 오직 몇개의 응용과 체계준위에서 SSI를 지원한다. 클라스터를 SMP로부터 MPP와 류사한 준위에로, 낮아 진 준위에서 SSI를 제공한다.

클라스터 혹은 MPP는 하나의 단일자원으로 사용될수 있다(실례로 하나의 거대한 워크스테이션). 한편 분산체계는 언제나 망령역에서 개별적콤퓨터들의 자립성으로 하여 다중체계망을 형성한다. 표 1-7에서 몇가지 체계에 대하여 다른 특징을 비교해 보자. 현재의 MPP와 분산체계는 최대로 수천개 마디로 되여 있다. 대부분 클라스터들은 수십개의 마디를 가지며 오직 몇개의 클라스터만 수백개를 넘는 마디들을 가진다.

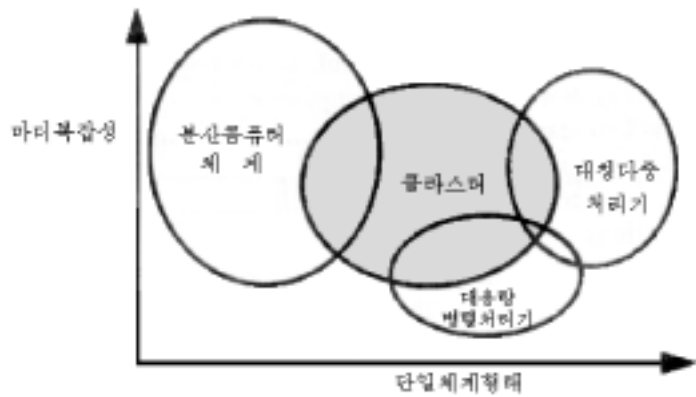



그림 1-8. 클라스터, MPPs, SMPs와 분산콤퓨터체계의 중복설계공간

MPP마디들은 자주 극소핵심부를 실행하며 한편 다른 구조에 있는 마디들은 완전한 OS를 실행한다. 분산체계에서 조작체계들은 보통 이질적이지만 다른것들은 동일한 OS를 리용한다. 마디간 통신은 때때로 MPP나 클라스터의 통보교환에 의하여 진행된다. 이와는 반대로 봉사기에서 공유된 파일들은 분산체계에서 주로 사용한다. SMP안에서 처리기들은 공유기억을 통하여 통신된다. 다만 SMP만은 하나의 단일한 주소공간을 가지고 있다. MPP는 DSM이 하드웨어와 결합될 때만 단일주소공간을 가진다.

SMP는 모든 처리기들에 대하여 단일실행대기렬을 가진다. 다중실행대기렬은 클라스터와 MPP마디들에서 사용된다. 그것들은 작업부하가 균등하게 조화된다. 분산체계에서 다중대기렬은 대부분 독립적이다. 분산체계는 표준통신망과 이질마디가동환경을 적응하게 해야 한다. 분산체계는 여러 체계들사이에 련계를 보장해 주어야 한다. 따라서 마디간 비밀안정성을 보장해야 한다. 여러 MPP나 클라스터는 이것들을 요구하지 않는다.

1. 4. 3. 클라스터의 우단점

클라스터의 개념은 많은 리익들을 주는것과 동시에 문제거리도 제기된다.

그가운데서 가장 중요한것은 리용가능성, 유용성, 확대가능성, 유용한 리용성, 성능대 비용비률이다. Ethernet를 통해서 워크스테이션의 한 묶음을 간단히 련결하고 그것들을 클라스터에 중첩하는것은 높은 적합가능성과 고성능체계를 구성하기 어렵다. 몇가지 기술(주로 쏘프트)은 이 포텐샬을 실현하는데 적절해야 한다. 5, 9, 10장에서 이 기술적문제를 자세히 고찰한다. 기본개념만을 이제 고찰하자.

리용가능성 클라스터의 마디들은 일반적으로 가동환경이므로 사용자들은 잘 알려

지고 잘 어울리는 환경에서 응용을 개발하고 실행할수 있다. 가동환경은 모든 강력한 워크스테이션프로그람작성 환경도구들을 가지고 있으며(순차적인) 수천개의 응용들이 실행되게 한다. 따라서 클라스터는 거대한 워크스테이션으로 볼수 있으며 이로부터 처리량을 증가시키고 다중순차사용자일감에 대한 응답시간을 감소시킨다.

병렬응용에서 클라스터는 통보교환 MPP보다 프로그람화하기가 더 어렵지 않다. 보충적으로 많은 자료기지판매자들은 자기의 상품들을 변경하여 클라스터에서 실행할수 있게 하였다. 그것들은 IBM DB2 Parallel Edition, Oracle, Sybase이다.

표 1-7 클라스터, MPP, SMP와 분산체계의 비교

체계특징	MPP	SMP	클라스터	분산체계
마디수 (N)	$O(100)$–$O(1000)$	$O(10)$ 혹은 less	$O(100)$ 혹은 less	$O(10)$–$O(1000)$
마디복잡성	Fine or medium grain	Medium or coarse grain	Medium grain	Wide range
마디사이통신	DSM에 대한 통보문넘기기와 공유변수	공유기억기	통보문넘기기	공유파일, RPC, 통보문넘기기
일감일정작성	Host에서 단일실행대기	단일실행대기	다중대기이지만 조절된다.	독립다중대기
SSI지원	부분적으로	언제나	요구	없다
마디 OS복사와 형태	N(마이크로핵심부)과 1 host OS (일치)	1 (일치)	N (동차요구)	N (heterogenous)
주소공간	다중(DSM이면 단일)	단일	다중	다중
마디사이보안	불필요	불필요	로출되면 요구	요구
소유	한개 조직	한개 조직	한개이상의 조직	많은 조직
망규약	비표준	비표준	표준 및 비표준	표준
체계유용성	낮은데서 중간	흔히 낮다	높은 리용가능성과 고장허용	중간
성능척도	처리량과 업무처리시간	업무처리시간	처리량과 업무처리시간	응답시간
SSI: 단일체계영상, RPC: 원격절차호출, DSM: 분산공유기억기				

유효성 유효성이란 용어는 체계가 생산적인 리용에 적합한 시간의 퍼센트를 의미한다. 일반적인 단일집적체계 즉 사용자대형콤퓨터, 고장유용성체계와 같은것들은 값이 비싸다. 전용화된 설계를 통하여 높은 유용성에 도달된다. 전용부분품들을 사용하는 대신에 클라스터는 값이 눅고 상업화된 부분품을 리용하여 높은 유용성을 제공하며 많은 여유를 준다.

- **처리기와 기억기** 클라스터는 다중기억기와 처리기부분요소들을 가진다. 하나가 실패할 때 다른것들은 여전히 클라스터의 동작을 계속하는데 사용될수 있다. 그와 대조적으로 SMP기계의 공유기억기가 실패하면 전체 체계는 정지된다.
- **디스크배렬** 클라스터는 여러개의 국부디스크들을 가진다. 따라서 하나의 고장은 전체 체계를 멈추지 않는다. 사실상 고장난 디스크를 가지는 마디는 여전히 기능을 수행하며 그것은 원격디스크에 의하여 기능을 수행한다.
- **조작체계** 클라스터는 다중 OS를 가지며 매개는 서로 다른 마디에서 동작한다. 하나의 체계화상이 기능을 중지할 때 다른 마디들은 여전히 동작할수 있다. 이것은 SMD와 대조적인것인데 SMP는 공유기억기에 있는 단일 OS만 가진다. 이 고장은 전체 체계를 정지시킨다.

클라스터의 유효성포텐샬을 실현하는 열쇠는 몇가지 쏘프트웨어기술이다. 이 기술은 역시 공유부분품을(실례로 호상련결) 고도로 유효하게 리용한다. 다른 한편 그것들은 고장의 원인으로 될수 있다. 이 기술들은 9장에서 론의될것이다.

확대가능한 성능 클라스터의 계산능력은 마디들이 보충될 때 증가할수 있다. 또한 클라스터확대가능성은 대량의 확대가능성이다. 이것은 SMP와 클라스터를 비교하면 잘 알아 볼수 있다. SMP들은 처리기확대가능성체계이고 한편 클라스터는 처리기, 기억기, 디스크, 지어 I/O장치까지 포함하여 많은 부분요소들로 확대한다. 성긴 결합이면 클라스터는 수백개마디로 확대될수 있다. 한편 그것은 수십개이상의 처리기를 가지는 SMP를 만드는것이 거의 불가능하다.

SMP에서 공유기억(기억모선)은 병목문제이다. 여러개의 순차프로그람들이 하나의 SMD우에서 실행될 때 이 프로그람들은 전체로 독립적이다. 같은 프로그람모임인 클라스터에서 실행될 때에는 기억기병목문제가 없다. 매개 프로그람은 하나의 마디에서 실행될수 있는데 그 마디는 국부기억을 리용한다.

이러한 응용들에서 클라스터들은 높은 전체 기억기대역너비와 감소된 기억지연시간을 제공할수 있다. 클라스터의 국부디스크들은 역시 거대한 디스크공간으로 집합된다. 그것은 집중화된 RAID디스크의 디스크공간을 쉽게 릉가할수 있다.

프로쎄스와 기억 I/O가능성을 강화하면 클라스터는 PVM 혹은 MPI와 같은 몇개의 개발된 쏘프트웨어를 사용하여 대규모의 문제들을 풀수 있다. 이것은 13장에서 론의된다.

클라스터들은 UP(단일처리기), SMP, MPP, 고장체계들을 그림 1-9에서처럼 확대가능성과 유효성의 관점에서 비교할수 있다. SMP들은 확대가능성이 높지 않다. 왜냐하면 경쟁모선과 집중공유기억기를 사용하기때문이다. 단일조작체계와 공유기억기는 포텐샬측면에서 두가지 고장이며 그것은 SMP의 유효성을 감소시킨다.

고장체계는 유효성이 높지만 그것들을 확대하려면 많은 비용이 요구된다. MPP는 그와 반대이다. 그것들은 쉽게 확대될수 있고 여전히 단일체계상을 보존한다. 클라스터들은 현재 중간정도에 있으며 앞으로 갱신되는데서 고성능화, 높은 유효성방향으로 갱신되고 있다.

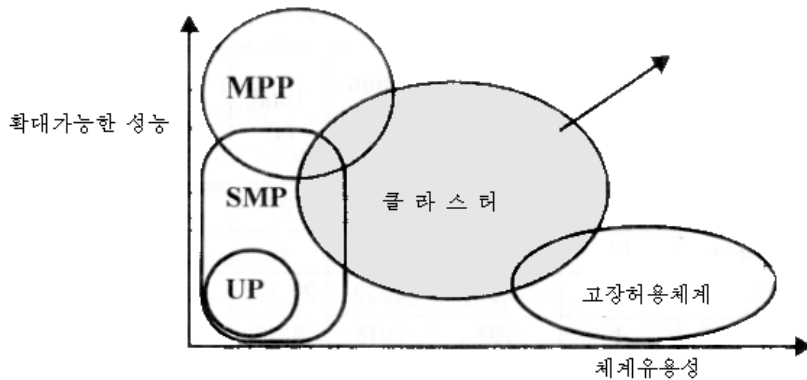



그림 1-9. 체계유용성에서 단일처리기, SMP, MPP,
클라스터와 그리고 고장허용체계의 비교

성능 대 비용률 클라스터는 비용효과성에서 많은 리득을 얻을수 있다. 전통적인 PVP고속콤퓨터와 MPP들은 수천만딸라로 쉽게 원가를 소비할수 있다. 다른 한편 같은 피크성능을 가지는 클라스터의 가격은 하나 혹은 두자리크기로 더 낮다. 클라스터들은 상품화된 요소들로 만들어 지는데 그것의 성능과 가격은 Moore의 법칙에 따른다. 그 법칙에 따르면 클라스터의 성능 대 비용률은 PVP와 MPP보다 더 빨리 늘어 나게 된다.

실례 1.8. 병렬콤퓨터와 클라스터의 성능 대 비용률

Berkeley NOW그룹의 두개의 병렬콤퓨터와 네개의 NOW구성의 성능 대 비용률을 비교하여 보면 표 1-8과 같다. 그 여섯개의 구성은 16처리기 Cray c90 PVP, 256마디 Intel

표 1-8 C90, Paragon 그리고 4개의 NOW체계의 성능/비용률

체계구성	ODE (s)	수송 (s)	입력/출력 (s)	전체 (s)	비용 ($M)	성능/비용률 (Mflop/s per $M)
Cray C90	7	4	16	27	30	44
Intel Paragon	12	24	10	46	10	78
NOW	4	23,340	4030	27,347	4	0.32
NOW+ATM	4	192	2015	2211	5	3.3
NOW+ATM+PIO	4	192	10	205	5	35
NOW+ATM+ PIO+AM	4	8	10	21	5	342

Paragon MPP, 256RS6000워크스테이션으로 이루어 진 망의 네가지 변종들이고 환경화학(대기화학)응용 Gator를 실행하였다.

프로그람은 세개의 기본단계를 가진다. 계산단계는 36Gflop를 수행하여 병렬로 상미

분방정식모임(ODEs)을 푸는 단계이다. 전송단계는 통신이 기본이다. I/O단계에서 3.9GB의 자료입구가 조절된다.

C90과 Paragon의 실행시간은 측정된 자료이다. 한편 NOW의 네 변종에 대한것들은 Berlekey그룹이 예상하였다. Ethernet를 사용하여 256개의 워크스테이션을 련결하고 통신규약으로 TCP/IP를 쓰면 PVM는 큰 체계를 만들어 내며 그것은 C90보다 1000배나 더 느리다. 그것의 성능대비용률은 138배나 더 낮다. 낮은 수준으로 되는 주되는 요인은 통신부가처리이며 그것은 C90에서 23340s와 4s이다.

Ethernet를 고대역너비 ATM교환과 교체하여 비용을 25%보충하면서 성능을 10배로 개선하였다. 병렬 I/O가능성을 실현하면 성능은 또 다른 준위에서 개선된다. 최종적으로 통신속도를 높이기 위하여 능동통보를 사용하면 실행시간은 한 자리크기로 감소된다. 적절하게 개선하면 클라스터 C90보다 더 빠르고 성능 대 비용률은 C90보다 7배 더 크다. 9장과 10장에서 하드웨어와 쏘프트웨어를 개선하여 어떻게 SSI클라스터를 개발하는가를 고찰한다.

1．5．확대가능성설계원리

확대가능한 고성능콤퓨터의 설계는 복잡한 공학적과정이다. 지난 20년동안 성공된 체계와 실패한 체계로부터 경험과 교훈에 기초하여 네개의 확대가능성설계원리가 창안되였다.

- 독립성의 원리
- 균형설계의 원리
- 확대가능성을 위한 설계원리
- 지연시간은페원리

이 원리들은 다음장들에서 중요한 역할을 논다. 이 부분에서 우리는 첫 세가지 원리를 설명한다. 지연시간은페문제는 5장에서 고찰된다.

1．5．1．독립성의 원리

이 원리는 체계의 구성요소들이 서로서로 독립되도록 해야 한다는것을 의미한다. 완전한 독립성을 달성할수 없으면 의존성은 될수록 작고 명백하게 해야 한다. 여기서 구성요소들은 하드와 쏘프트부분품요소들이다.

한가지 명백한 리득은 독립적인 확대가능성이다. 즉 우리는 체계를 하나의 구성요소들은 다른것들과 독립적으로 개선하는 방법으로 한 차원에 따라 체계를 확대할수 있다. 이것을 증가되는 **확대성**이라고 부른다(증분확대라고 부른다.).

사용자가 마디들을 첨부하여 병렬콤퓨터크기를 확대하려면 조작체계나 프로그람작성환경을 한급 높이 갱신할것을 요구하지 않는다. 처리기가 다음세대로 갱신되면 체계는 나머지구성요소를 갱신함이 없이 높은 성능에서 동작할수 있다. 또 하나의 리득은 종이

다른것을 확대가능하게 하는것이다. 구성요소가 특정한 구조와 체계에 매이지 않기때문에 그것은 다른 체계를 만들 때에 사용될수 있다. 이것은 결과적으로 비용을 절약하게 한다. 리상적으로 구성요소는 다음의 특징을 가지고 상품화되고 있다.

- 나머지체계에 대한 표준대면부를 가지는 열린 구성방식을 가진다.
- 즉시 구입할수 있는 상품이여야 하며 더 좋기는 특허가 아니고 누구나 가질수 있어야 한다.
- 여러 판매자들이 열린시장에서 대량으로 판매해야 한다.
- 상대적으로 잘 어울리며 충분히 오랜 시간동안 많은 사람들이 사용하고 잘 검열수정해야 한다.
- 따라서 상품화된 구성요소들은 낮은 비용과 높은 유효성 그리고 신뢰성을 가져야 한다.

아래에 독립성의 원리를 구체적인 실례를 들면서 설명한다.

- 알고리듬은 구성방식과 독립이여야 한다.
- 응용은 가동환경과 독립이여야 한다.
- 프로그람작성언어는 기계와 독립이여야 한다.
- 마디는 망독립적이고 망대면부는 망위상독립적이여야 한다.

실례 1.9. 인터네트와 IBM SP2을 개발하는데서 독립성

인터네트의 성공은 독립성의 원리의 리익에 대한 실례이다. 인터네트는 호스트호상련결하드와 응용쏘프트에 무관계하다. 서로 다른 형태의 호스트들이 련결될 때 그것은 PC로부터 각이한 판매자들로부터 구입한 고속콤퓨터에 이르기까지 포괄한다. 호상련결된 하드는 Ethernet, ICCI 기타 등등일수 있다. 사용자들은 Web를 리용할 때 각이한 쏘프트를 쓸수 있다.

IBM SP2설계는 독립성의 원리를 도입하였다. 마디구조는 각이한 통신구조(실례로 Ethernet나 HPS)를 쓸수 있게 하였다. 통신구조는 통신하드와 독립적이며 Ethernet 또는 HPS에서 표준 IP규약 또는 IBM이 독점하고 있는 사용자공간규약이 사용될수 있다.

실례 1.10. MPI와 초립방콤퓨터

통보문넘기기대면부(MPI)는 적은 언어특징을 사용하는 좋은 실례이다. MPI는 서로 직교하는 네가지 주요개념에 토대하고 있다. 즉 자료형, 통신동작, 통신기, 가상위상이다.

네가지 임의의 조합은 타당하다. 이 직교독립성은 곱하기연산을 요구한다. 몇개의 간단한 개념들이 서로 조합될 때 더 많은 가능성을 제공한다.

초기의 초립방콤퓨터들에 대하여 개발된 많은 병렬알고리듬들은 분명히 호상접속의 초립방위상을 사용하였다. 그것은 그물망집합적인 체계에서 잘 수행되지 않을수 있다. 현재 MPP들은 호상련결위상과 독립인 통신알고리듬이라는것을 쓰고 있다. 좋은 실례는 IBM SP체계에서(집중통신이다[59] .) 독립성의 원리를 실현하는데 두가지 일반적인 기술

이 리용되였다. 즉 실행과 구성방식의 분리, 표준구성요소의 리용기술을 리용한다.

구성방식과 실행 구성방식의 개념은 실행개념과 분리되여야 한다. 구성방식은 콤퓨터체계 또는 체계구성요소들로 이루어 진 모형이다. 실행은 모형의 특정한 현실화이다. 구성방식모형은 사용자와 설계자들이 사용한다. 구성방식은 각이한 실현을 가질수 있고 그 실현은 각이한 성능들을 가지지만 같은 기능성을 실현한다.

구성방식개념은 오래전의것이며 그것은 IBM 360의 설계에서 처음으로 제기되고 성공적으로 응용되였다. 구성방식개념에 가까운것은 가족집단개념이다. 가족집단의 모든 성원들은 같은 구조를 공유한다. 그러나 각이한 성원들은 각이한 실행과 각이한 성능 대 비용요구에 적합한 구성을 사용할수 있다. 또 하나의 널리 알려 진 관련개념은 열린 구성방식(또는 열린 체계)개념이다. 그것은 구성방식의 소유자가(보통 판매자) 제3자들이 구성방식을 만들 때 요소들을 변경하고 그것을 다시 제조하며 가동할수 있도록 한다는것을 의미한다. IBM PC의 성공은 참으로 기술적으로 강력하고 상업적으로 실용적인것이였다.

표준부분품요소의 리용 구성방식에 대한 론의는 벌써 표준구성요소의 사용에 대한 중요성을 암시해 준다. 두가지 형태의 표준이 있다. 즉

- 첫번째 형태는 공업규격ㄴ(역시 de facto표준이라고 부른다.)이다. 그것은 보통 회사에 의하여 개척되였고 다음 말단사용자들에 의하여 사용되고 있으며 대부분의 공업들에서 도입되였다.
- 두번째 형태의 표준은 민족 또는 국제규격화기구 즉 International Standards Organization(ISO), American National Standards Institute(ANSI), IEEE표준위원회 등에 의해서 창안되였다.

주의

독립성의 원리를 적용할 때 다음과 같은 두가지를 주의해야 한다.

- 첫째로, 병렬콤퓨터들은 계산기술의 첨단에 있다. 이러한 체계에는 잘 알려 져 있지 않는 중요한 구성요소/기술이 있으므로 완전히 표준은 아니다.
- 둘째로, 하나 또는 몇개의 구성요소들을 단독으로 간단히 확대하여 효과적인 체계를 만들수 없다.

실례로 처리기속도가 빠르다. 느린 기억기모듈 혹은 느린 통신부분체계를 리용하지 말아야 한다. 모든 부분체계들도 균형화된 체계로 설계해야 하는데 다음의 항목에서 론의한다.

1. 5. 2. 균형설계의 원리

이 원리는 성능병목을 될수록 최소화해야 한다는것을 의미한다. 불균형체계를 설계하는것은 피해야 한다. 비록 나머지요소가 빠르다고 해도 느린 요소는 전체 체계의 성능을 떨군다.

더우기 단일점고장을 피해야 한다. 다시 말하면 한 요소의 고장이 전체 체계를 정지시키는것을 피해야 한다. 몇가지 리론적 및 실험적결과들에 의하여 균형적으로 설계할수 있다.

Amdahl의 법칙 응용프로그람이 두 형태의 계산구조로 그룹화된다고 하자. 즉 부분 X와 부분 Y이다. 두 부분은 각각 전체 실행시간의 X%, Y%를 취한다.

X%	Y%

X가 개선되여 n배 더 빨리 실행되면 속도증가 S는 다음과 같이 표시된다.

$$S = \text{초기시간/개선된 시간} = \frac{1}{(X/n)\% + Y\%} \to \frac{1}{Y} \qquad (1.5)$$

이 방정식은 Amdahl의 법칙으로 알려 져 있는데 다음과 같은 의미를 가진다.

- 보다 큰 부분 X를 최량화해야 한다. 대부분 경우에 속도를 증가시킨다.
- 가장 좋은 속도증가웃한계로서 1/Y값을 가진다.
- 느린 부분 Y를 **병목**이라고 부른다. Y를 최소화해야 한다. Y를 가능한 작게 해야 한다.

Amdahl의 규칙 이것은 역시 Amdahl의 다른 법칙으로 알려 져 있다. 그것은 처리속도가 기억기용량과 I/O속도에 대하여 균형화되여야 한다는것을 의미한다. 보다 명확하게는 1MB의 기억용량과 1Mb/s의 I/O비률은 초당 백만개명령(MIPS)계산속도로 균형을 이루어야 한다는것이다.

Amdahl의 규칙은 경험적이다. 이처럼 그것은 반드시 피크속도에 한해서는 관측되지 않는다. 현재 처리기들은 약 400~1200Mi/s의 범위에서 피크속도를 가지지만 많은 기계들은 처리기당 400MB이상의 국부기억기를 설치하지 않고 있다. Amdahl의 규칙은 피크속도대신에 지속계산속도(실례로 일관하게 Mflop/s)가 쓰일 때 최근 체계들에서 더 많이 관측된다.

실례 1.11. PetaFLOPS프로젝트

PetaFLOPS프로젝트[587]로부터 최근에 예측되는 광범한 과학/공학모의문제들에서 기억기요구(GB)와 속도요구(Gflop/s)는 다음의 관계를 가진다.

$$기억 \sim 속도^{3/4} \quad (1.6)$$

따라서 약 30TB의 기억기크기는 Pflop/s기계에 적당하다. 1Pflop/s = 1000000 Gflop/s이다.

실례 1.12. I/O와 검사점문제들

I/O속도요구를 리해하기 위하여 검사문제를 고찰한다. 체계는 주기적으로 기억기내용을 디스크에 옮기여 체계가 갑자기 멎는 경우에 사용자들이 작업을 제일 처음부터 시작하는 대신 마지막검사점에서부터 시작할수 있게 할 필요가 있다. 90s내에 끝나도록 덤핑하려고 한다고 가정하자. 그러면 1MB기억기에 대하여 디스크대역너비를 1/90MB/s=0.9MB/s로 해야 한다. 그것은 Amdahl의 규칙에 가깝다. 보다 큰 체계에 대하여 검사점시간은 더 길다. 덤핑하는 시간을 900s라고 하자. 그러면 1GB기억을 가지는 기계에 대하여 디스크대역너비를 1000/900=1.1MB/s로 하여야 한다. 100GB기억에 대하여 디스크 I/O요구는 100MB/s이상으로 증가한다.

50%규칙 식 (1.4)로부터 병렬프로그람의 성능은 부하불균형, 병렬성부가처리, 통신시동부가처리바이트당 통신부가처리에 의하여 감소될수 있다. 경험에서 얻은 법칙은 처리인자가운데서 매개가 50%이하의 인자에 의하여 성능을 낮출수 있으면 병렬체계는 균형된다는것이다. 이 규칙을 사용하여 다양한 부가처리인자들에 관한 기대되는 한계를 평가할수 있다. 이것은 3장에서 취급될것이다.

실례로 표 1-9는 통신시동부가처리 t_0에 대하여 각이한 속도조건밑에서 요구되는 값을 렬거한다. 현재 통보문넘기기콤퓨터들은 시동부가처리가 너무 커서 미세하게 작은 병렬응용을 지원하지 못한다.

일부 Cray벡토르고속콤퓨터(PVP)를 제외하고 점대점통신에서는 오직 2μs의 작은 t_0을 가진다.

표 1-9 요구되는 통신, 시동, 부가처리값

Grain 크기 w	단일마디 속도 P_1	t_0은 아래값보다 작아야 한다
100 flop (Fine Grain)	4 Mflop/s	25 μs
	20 Mflop/s	5 μs
	100 Mflop/s	1 μs
1000 flop (Medium Grain)	10 Mflop/s	100 μs
	50 Mflop/s	20 μs
	250 Mflop/s	4 μs
10000 flop (Coarse Grain)	10 Mflop/s	1000 μs
	100 Mflop/s	100 μs
	1000 Mflop/s	10 μs

통보문크기 α*w 또한 립도 w가 크면 대역너비 $r_\infty=1/t_c$는 보다 중요한 인자로 된다. 표 1-10은 속도 P_1와 통신 대 계산률 α를 각이하게 가정한데 기초하여 r_∞의 기대되는 값을 렬거한다.

실례 1.13. PDE풀기에서 격자

수값병렬편미방(PEE)풀기는 2차원문제들을 위해서 다음의 도식을 사용한다. 자료령역은 N*N자료점을 가지는 2D격자이다. 매 자료점은 XB의 기억을 요구하며 전체 기억요구는 N^2*B이다. 알고리듬은 일정한(실레로 10000) 시간단계를 수행한다. 매 단계에서 격자점은 Y+10p계산을 요구하며 그것의 네개의 린접점들에 접근한다.

단일처리기를 리용하는 실행시간은

$$T_1 = YN^2 t_f \tag{1.7}$$

이다.

표 1-10 요구되는 통신, 대역너비값

통신대계산률 α	단일매듭 속도 P_1	r_∞은 아래값보다 커야 한다.
0.01 B/flop	4 Mflop/s	0.04 MB/s
	20 Mflop/s	0.2 MB/s
	100 Mflop/s	1 MB/s
0.1 B/flop	10 Mflop/s	1 MB/s
	50 Mflop/s	5 MB/s
	250 Mflop/s	25 MB/s
1 B/flop	10 Mflop/s	10 MB/s
	100 Mflop/s	100 MB/s
	1000 Mflop/s	1000 MB/s

n-처리기 MPP에서 자료령역은 n개의 평방구역으로 나눌수 있으며 매개 구역은 N^2/n개의 격자점을 포함하는 $(N/\sqrt{n})*(N/\sqrt{n})$부분령역이다. 매 시간단계에서 처리기당 계산작업부하는 YN^2/n이다. 매 처리기는 네개의 매 린접점으로부터 $XN\sqrt{n}B$이 얻어진다.

따라서 n-처리기기계에서 실행시간은 근사적으로

$$T_n = \frac{YN^2}{n} + 8\left(t_0 + \frac{NXt_c}{\sqrt{n}}\right) \tag{1.8}$$

이다. 상수인수 8은 매 처리기가 네개의 매 린접점과 통보를 주고받기해야 하는데로부터 나온다. 속도증가인자는 다음과 같다.

$$S = \frac{T_1}{T_n} = \frac{n}{1 + \frac{8t_0}{YN^2 t_f} + \frac{8Xt_c}{YNt_f\sqrt{n}}} \quad (1.9)$$

그림 1-10에서 네가지 서로 다른 병렬콤퓨터들에 대한 속도증가곡선을 보여 준다.

(1) 첫째 기계는 t_0=555μs와 r_∞=1MB/s를 가지는 느린 통신망으로 련결된 50Mflop/s 처리기들을 가지고 있다. 많은 점대점통신을 위해서 Ethernet와 호환성을 가진다. 문제크기가 N=102일 때 기계 A는 아주 작은 속도증가를 보여 주는데 (그림 1-10의 낮은 부분에 곡선으로 표시된다.) 128개 처리기에 대하여 10이하이다.

(2) 그러나 속도증가는 문제크기 N이 8k로 8배 증가하면 급격히 개선된다(그림 1-10에서 ◆으로 표시된다.) 이것은 립도 w를 증가시키며 통신 대 계산비 α를 감소시킨다.

(3) 셋째 기계(그림 1-10에서 △으로 표시된)는 100-Mflop/s처리기들을(첫 기계보다 2배 빠르다) 가지는데 그것은 같은 망에 의해서 련결된다. 이 불균형체계설계로 하여 속도증가는 N=809에 대하여 떨어 진다.

(4) 마지막기계(그림 1-10의 꼭대기에 곡선으로 표시된)는 기계 B보다 더 균형을 이루며 그것은 t_0=46μs, r_∞=35MB/s를 가지는 IBM SP2과 류사한 망으로 련결된 100Mflop/s처리기들을 가지고 있다. 이 균형화된 설계는 4개의 병렬콤퓨터들가운데서 제일 좋은 성능을 가진다.

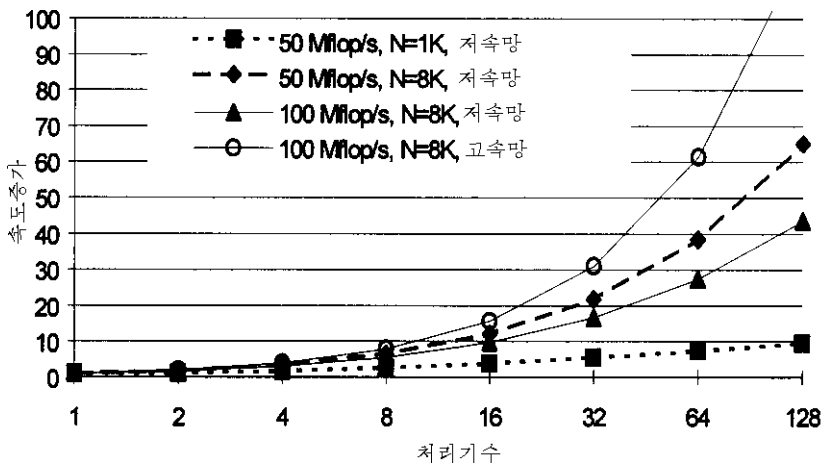



그림 1-10. 4개 병렬콤퓨터에서 2D PDE풀기의 속도증가

1. 5. 3. 확대가능성을 위한 설계

이 원리는 확대가능한 체계를 설계할 때 확대가능성은 후에 생각해 볼것으로 하지 말고 시작부터 주되는 목표로 고찰해야 한다는것을 의미한다. 체계는 고성능을 제공하기 위해 확대될수 있거나 비용효과성을 더 크게 하기 위하여 축소할수 있도록 준비되여야 한다. 확대가능성을 설계하는데 많이 사용하는 두가지 방법론은 여유설계와 역호환성이다.

여유설계 여유설계기술을 사용하면서 체계는 현 세대 체계의 최소요구를 겨우 만족하지 않도록 설계한다. 설계는 앞으로 체계를 확대할 때 개선되는것을 예견하는 보충적인 특징들을 포함해야 한다. 이 특징들은 현재 체계에서 의의가 없지만 그것은 앞으로 개선되는 체계에로 원활하게 이행할수 있게 한다.

실례 1.14. 현대적인 처리기설계에서 주소화공간

처리기를 설계하는데서 가장 중요하게 고려해야 할것의 하나는 처리기주소공간 크기이다. 바꾸어 말하면 처리기가 직접 접근할수 있는 byte위치의 수이다. Gordon Bell의 말을 인용하면 작은 주소공간은 후에 쉽게 고쳐 지지 않는 구성방식설계의 오유이다.

현대적인 처리기들은 64bit주소공간 또는 $2^{64}=11.8\times10^{19}$B을 리용한다. 이 거대한 주소공간은 오직 32b(4-GB)를 지원하는 UNIX에 의해서는 충분하게 리용되지 않는다. 이 주소공간에서의 여유설계는 조작체계가 64BUNIX에로 확대될 때 아주 쉽게 넘어 가게 한다.

공간

반례로서 초기 IBM PC에서 사용된 Intel 8086/8088주소처리기는 20bit로 한정된 1MB주소화공간을 가지고 있었다. DOS는 Kernel과 사용자쏘프트 640KB의 한계를 설정한다. 이 640KB한계는 콤파일러작성자들과 응용쏘프트개발자들에게 많은 문제들을 제기한다.

DOS프로그람이 640KB한계를 넘을수 없기때문에 쏘프트설계자들은 정교한 기술(실례로 고대역기억기, 확장기억기, 연장기억기)을 반영하여 다음세대처리기(Intel 286, 386, 486, Pentium Pro)에서 제공되는 거대한 주소공간에 대한 우월성을 가지게 해야 한다.

실례 1.15. IBM RS6000SMP봉사기들에서 여유설계

IBM RS6000SMPl는 세대확장가능성을 위하여 여유설계되여 있다. 실례로 SMP의 첫 세대는 PowerPC601처리기에 토대하고 있다. 그러나 체계 나머지부분품요소기억 I/O, 전원, 선풍기, 시계회로들은 PowerPC604와 620처리기 두 세대에 쓸수 있게 설계되고 있다.

스위치의 매 처리기포구는 600MB/s의 대역너비를 가지며 601에서 요구하는것보다 더 크다. 이 여유설계특징들은 SMP체계의 미래세대들에서 간단히 처리의 수준을 높이는 것에 의해서 확대하기 쉽게 한다.

역호환성 여유설계기술은 현재체계를 설계할 때 체계를 확대하기 위한 요구들을 고찰해야 한다는것을 의미한다. 역가능성이라고 부르는 기술은 하드와 쏘프트부분요소들을 설계하는데서 체계를 축소하는 요구를 고찰해야 한다. 확대된 요소들은 이전 또는 축소된 체계에서도 사용할수 있어야 한다. 실례로

- 새로운 처리기는 이전 처리기들에서 2진코드들을 실행할수 있어야 한다.
- *n*마디우에서 실행되게 설계된 병렬프로그람은 단일마디(*n*=1)우에서 축소된 입구자료모임을 가지고 실행될수 있어야 한다.
- 고속콤퓨터를 위한 프로그람은 워크스테이션우에서 실행될수 있어야 한다. 또한 입구자료모임은 워크스테이션의 제한된 주기억에 맞아야 한다.
- 조작체계의 새로운 변종은 쓸모 없는 모든것들을 제외하고 모든 이전 기능들을 보존해야 하는데 그것은 명백하게 문서화되여야 한다.

실례 1.16. 처리기에서 자료모선너비의 호환성

현대 처리기들의 CPU모선은 64bit 자료모선을 가진다. 어떤 처리기들은 역호환성을 지원하는데 그것은 64bit처리기가 보다 작은 자료모선너비들(32, 16, 8bit)을 가지는 주기판에서 사용되도록 한다.

이 역호환성은 몇개의 조종판들을 설정하여 달성한다. 처리기가 기억기읽기를 할 때 두개의 모선주기(하나는 주소에 보내고 다른것은 기억기로부터 자료를 읽는다.)를 요구한다고 하자. 읽기가 8bit주기판에서 실현되면 처리기의 설정은 자동적으로 9개의 모선주기안에서 기억기로부터 1B자료를 읽는다.

실례 1.17. IBM SP체계들에서 병렬조작환경

IBM병렬조작환경(POE)은 IBM ScalableParallel콤퓨터들(SP1과 SP2)에 대하여 설계되였다. 여기서 사용자응용은 여러 SP마디우에서 실행된다. POE는 역가능성을 다음과 같이 지원한다.

- POE는 한마디에서 완전히 실행될수 있고 그것은 여러 리용자과제들이 한마디우에서 실행되게 한다.
- 그 마디는 SP마디가 아닐수 있다. 어떤 RS6000워크스테이션일수 있다.
- POE는 IBM UINX (AIX)의 꼭대기에서 실현된다. 따라서 현재 UNIX프로그람들은 POE에서 리용할수 있다.

경고 역호환성은 이전 체계의 모든 특성들이 유지되여야 한다는것을 의미하지 않는다. 쓸모 없는 특징들은 제거해야 한다. 여유설계기술을 적용하는데서 초래되는 비용을 고려해야 하며 앞으로에 대하여 너무 많이 예견하지 말아야 한다.

때때로 여유설계는 실제적으로 전체 개발과 생산비용을 감소시킨다. 이것은 Intel의 다음실례로부터 설명된다.

실례 : 1.18 Intel 486 DX와 486 SX

Intel은 80486극소형처리기를 발표할 때 두가지 모형 즉 486DX와 486SX를 제기하였다. 내부적으로 두 소편은 동일하다. 차이는 오직 DX소편의 류점단(FPU)이 SX소편에서 기능하지 못하게 되여 있고 류점능력을 요구하지 않는 사용자들에 대해서 판매되였다는 것이다.

FPU에 대한 소편령역이 SX소편우에서 리용하지 못한다고 해도 이 여유설계로 하여 Intel은 두 소편을 설계하지 않아도 된다. 두 소편의 결합판매는 설계와 생산비용이 대단히 줄어 들게 하며 따라서 리윤이 크다는것을 보여 주었다.

1．6．참고문헌주해와 련습문제

병렬처리기들은 역시 Flynn[245]와 Hennessy, Datterson[305]에서 취급된다. Hwang[327], Lenoski와 Weber[406], Culler[181]들은 확대가능한 다중처리기를 취급하고 각이한 령역에 대해서 취급되는 내용을 강조하고 있다. 체계확대가능성의 론의는 Hill[308], Rattner[513], Smith[569]가 준다. Ni[464]병렬콤퓨터를 층별로 분류한다. Almasi와 Gottieb[20]은 1994년까지 고도의 병렬계산체계들을 넓은 범위에서 포괄한다.

콤퓨터로 이루어 진 클라스터의 기본개념은 Pfister[497]에서 소개된다. 망기초클라스터계산에 관해서는 두가지 문제가 Journal of Parallel and Distributed computing (1997년 2월, 6월)에서 출현하였다. 병렬 및 클라스터계산에서 최근 앞선 내용은 역시 Bell과 Gray[71], Davy와 Dew[190]에서 론의된다.

확대가능설계원리들은 Hwang과 Xu[333], Herris와 Tophaw[297]의 론문에서 취급된다.

PRAM모형은 Fortane과 Wyllie[240]에 의하여 제안되였다. BSP모형은 Valian [623]에서 서술된다. 단계병렬모형은 Xn과 Hwang[657]에서 주었다.

Amdahl의 법칙은 처음에 [29]에서 주었다.

Gordon Bell은 현대기술에 기초하여 고속콤퓨터와 MPP들을 평가하였다. [67]~[71]확대가능성콤퓨터가동환경들은 IBM[335]~[341], DEC[191]~[198] Cray Research[3]과 [174], SGI[547]~[550], Sun Microsystems[198]~[600], Microsoft’s Wolfpack[573], Convex/HP[165]와 [166]들에 제기되였다. 이 체계들은 다음장들에서 연구될것이다.

문 제

문제 1.1 확대가능성에 관한 다음 용어들을 정의하고 구별하시오.

- 기계크기에 관한 확대가능성
- 문제크기에 관한 확대가능성
- 자원확대가능성
- 세대확대가능성
- 이종확대가능성

문제 1.2 다음의 추상병렬모형들을 비교하시오. 그것들의 차이, 상대적우점, 생활력있는 병렬콤퓨터와 응용을 모형화하기 위한 한도들을 론하고 평가하시오.

- PRAM모형
- BSP모형
- 단계병렬모형

문제 1.3 다음 다섯개의 병렬구조에서 매개의 고유한 특징을 두개 이야기하시오. 고유한 특징은 임의의 다른 모형들에서 볼수 없다는것을 론하시오.

- 병렬벡토르처리기(PVP)
- 대칭다중처리기(SMP)
- 대용량병렬처리기(MPP)
- 클라스터워크스테이션(COW)
- 분산공유기억(DSM)기계

문제 1.4 현재의 병렬콤퓨터들에서 병렬실행방식에 대한 다음의 질문에 대답하시오.

- SIMD,MIMD,SDMD, MDMD실행방식의 우점과 결점은 무엇인가
- SIMD콤퓨터우에서 MIMD병렬응용은 어떻게 실현되는가. 도식을 설명하는데 구체적인 실례를 사용하시오.
- 오직 SDMD방식을 지원하는 콤퓨터우에서 MDMD병렬응용을 어떻게 실현하는가. 도식설명에서 구체적인 실례를 사용하시오.

문제 1.5 독립성의 원리에 관하여

- 충분히 확대가능한 체계에서 가능한 구조특징의 독립쌍을 렬거하시오.
- 같은 구조의 다중실현으로부터 구조의 개념을 설명하고 어떻게 분리하는지 실례를 들어 설명하시오.

문제 1.6 확대가능병렬 및 클라스터계산과 관련하여 다음의 성능을 정의하시오.

- 병렬실행시간
- 작업부하와 지속적인 속도
- 병렬성과 호상작용부가처리
- 점대점 및 집합적통신

문제 1.7 다중처리기, 다중콤퓨터, 콤퓨터클라스터에 관하여

- 다중처리기와 다중콤퓨터를 자원공유, 구조, 처리기사이 통신에 기초하여 구별하시오.
- UMA, NUMA, COMA, DSM, NORMA기억모형사이의 차이를 설명하시오.
- 자료콤퓨터로 이루어 진 고전적인 망에서는 볼수 없는 클라스터의 보충적인 기능특징은 무엇인가?
- 일반적인 SMP봉사기에 대한 클라스터체계의 우월성은 무엇인가?

문제 1.8 PRAM모형을 써서 현재의 병렬콤퓨터들을 모형화하시오.

- 어떤 PRAM변종이 SIMD기계를 더 좋게 모형화할수 있는가? 어떻게 모형화하는가 설명하시오.
- 어떤 PRAM변종이 공유기억 MIMD기계를 더 잘 모형화할수 있는가?
- 오늘 만들어 져 있는 대부분의 병렬콤퓨터들을 모형화하는데서 PRAM의 불충분한 점을 평가하여 보시오.

문제 1.9 병렬콤퓨터의 속도증가성능을 고찰하시오.

- Amdahl의 법칙의 공식을 이야기하고 그것의 의미를 대략적으로 설명하시오.
- Amdahl의 규칙이란 무엇인가. 병렬체계설계에 어떻게 적용하는지 설명하시오.
- 자주 사용하는 콤퓨터체계 두개를 검열하고 Amdahl의 규칙이 나오는지 보시오.

문제 1.10 확대가능성 또는 클라스터극소처리기와 다중콤퓨터체계들을 만드는데서 다음설계방법들을 사용하시오.

- 두개의 실례설계를 써서 여유설계가 어떻게 확대가능성을 강화할수 있는가 설명하시오.
- 확대가능콤퓨터를 만드는데서 역가능성은 왜 중요한가?
- 고도로 유효한 클라스터체계에서 검사점성능은 무엇인가?
- 다중콤퓨터클라스터들과 분산콤퓨터들에서 마디복잡성, 마디조작체계, 마디사이통신, 일정작성기, SSI에 대한 지원, 체계유효성, 안전조종의 차이를 비교하시오.

제2장. 병렬프로그람작성의 기초

병렬프로그람작성은 주어 진 알고리듬으로부터 병렬프로그람을 구성하는 작업이다. 그것은 알고리듬설계자와 콤퓨터체계구성방식설계자들의 폭 넓은 호상작용을 연구한다. 이것에 대하여 H. J. Siegel 등[256]이 강조하였다.

일반적인 고성능계산분야에서 두개의 가장 중요한 분야는 구성방식과 알고리듬이다. 그러나 그것들사이의 대면부도 역시 중요한 문제이다.

여기서 우리는 병렬프로그람작성의 상태를 고찰하고 병렬성, 프로쎄스호상작용, 병렬프로그람들의 기본성질들에 관한 표현들을 검토한다. 모든 실례코드들에서 다르게 표시되지 않는한 *는 곱하기기호를 표시한다. +는 하나씩 증가한다는것을 의미하며 x% 7는 x mod 7을 의미하고 x == y는 조건검사 즉 c표기법에서 <x는 y와 같다>를 가리킨다. 표기법 x : =5는 변수 x에 5를 배정하는것을 의미한다.

2. 1. 병렬프로그람작성의 개요

병렬프로그람작성이 유감스러운 상태에 있다는것은 일반적으로 일치한 의견이다.

- 병렬쏘프트웨어개발은 병렬하드웨어가 앞선데 비하여 멀리 뒤떨어 져 있다.
- 순차형쏘프트웨어와 비교해 보면 오늘의 병렬체계쏘프트웨어와 응용쏘프트웨어는 량적으로 적고 기능적으로 원시적이다.

이러한 경향은 여러가지 리유로 하여 계속될것이며 그것은 앞으로도 계속 론의될 전망이 보이고 있다. 여기서는 병렬하드웨어와 병렬쏘프트웨어사이의 간격을 좁히는 최근 발전에 대해서 고찰한다.

2. 1. 1. 병렬프로그람작성이 왜 어려운가

병렬쏘프트가 뒤떨어 진 주되는 리유는 병렬프로그람작성이 순차프로그람작성보다 훨씬 복잡하고 지적인 과정이라는것이다. 그것은 첫째로, 병렬프로그람은 순차프로그람작성을 포함하고 있기때문이다. 그것은 순차프로그람작성의 모든 문제들을 포함하며 나머지 문제는 보다 지적으로 해결해야 할 대단히 많은 문제들을 더 가지고 있기때문이다.

둘째로, 순차프로그람작성에는 오직 하나의 기초모형(Von Neumann)이 있지만 한편 병렬프로그람작성에는 많은 각이한 모형들이 있다. 셋째로, 콤파일러, 프로필리와 같은 쏘프트환경도구들은 순차프로그람개발에서 훨씬 앞서고 있다.

넷째로, 병렬프로그람작성보다 순차프로그람작성에서 보다 많은 사람들이 실제로 체험하고 있으며 또한 아주 오랜 기간 사용하고 있다. 다른 말로 말하여 순차프로그람작성은 보다 잘 어울리며 축적된 지식의 거대한 기지를 가지고 있다는것이다. 그 기지는 과거에 배운 과정들과 오유찾기를 포함하고 있다. 병렬프로그람작성을 그림 2-1의 실례에서와 같이 사용자의 관점에서 비교해 보자.

순차프로그람작성 사용자에게 응용프로그람이 요구된다고 하자. 순차기계에서 그러한 응용이 존재하고 수정된다. 더 좋기는 원천코드(실례로 C 또는 포트란)가 유효할것이다. 사용자는 오직 그 코드를 다시 콤파일하여 목표기계에서 그것을 실행한다. 응용이 개발되자면 사용자는 목적에 부합될수 있는 현재알고리듬을 찾아야 할것이다. 알고리듬이 없으면 사용자는 몇가지 프로그람작성도구들의 도움을 받아야 한다.

첫째로, 오래동안 수립되였고[16] 알고리듬을 설계하는데서 사용자를 안내할수 있는 몇가지 알고리듬견본이 있다. 둘째로, 꼭 하나의 기초모형 Von Neuman[271]이 있어서 종합적으로 알고리듬적견본들, 프로그람작성언어들 그리고 콤퓨터가동환경들을 지원한다. 마지막으로 많은 순차언어들이 제안된것이다. 따라서 적은 수의 표준언어들이 있어서 즉 Fortran은 과학계산에서 지배적이며 코볼과 4세대언어들(4GL)은 자료처리에서 널리 쓰이며 C는 어디서나 사용된다.

순차프로그람작성도구는 일반적이다. 실례로 C언어는 모든 알고리듬견본들을 지원할수 있고 모든 계산가능한 함수들을 규정할수 있다. C프로그람이 개발되였다면 임의의 순차콤퓨터에서 리용할수 있다. 다른 말로 말하면 순차프로그람작성은 이질적확대가능하다(1.2.3). 순차프로그람작성도구들은 콤퓨터가 진화되는 많은 세대들에서 일정하다(다시 말하면 세대확대가능하다.).

Fortran 77은 1977년에 표준화되였고 오늘 여전히 널리 쓰이고 있다. 오늘 사용되는 C언어는 20년전에 발명한 원시변종에 비하면 약간 변화되였다. Von Neuman모형은 40년이상 많은 변경없이 사용되여 왔다. 그것의 일반성과 안정성으로 하여 도구들은 널리 리용된다. 그것들은 각이한 콤퓨터가동환경에 있는 각이한 응용을 가지고 오래동안 검사되여 왔다. 사람들은 그것들을 어떻게 잘 사용하며 오유을 피하겠는가를 연구하고 있다.

병렬프로그람작성 병렬프로그람작성은 모든 측면에서 미약하다. 병렬코드가 어떤 기대되는 응용에 대하여 존재한다는 가망은 없다. 지어 그렇다고 하면 사용자의 병렬콤퓨터(실례로 MPP)에 대하여 사용할수 없다. 왜냐하면 병렬코드는 다른 구성방식에 대하여(실례로 SMP봉사기) 원시적으로 작성되기때문이다.

병렬프로그람작성은 성숙되고 일반적인 안정한 도구들의 지원을 가지지 못한다. 병렬알고리듬견본들은 여전히 잘 리해되지 않으며 널리 도입되지도 않고 있다.

단일한 일반적인 기계모형대신에 두 준위의 모형이 있는 매개 준위에는 많은 각이한 모형들이 많다.

프로그람작성모형은 프로그람작성자가 병렬프로그람을 작성할 때 보고 사용하는것이다. 특정모형은 특정한 병렬콤퓨터가동환경에 의하여 제공되는 가장 낮은 준위의 사용자가 볼수 있는 프로그람작성모형이다. 다른 프로그람작성모형들은 이 특정모형에서 실현될수 있다. 실례로 SMP인 SGI Power Challenge우에서 특정모형은 공유변수모형(실례로 SGI Power6)이다. 그러나 자료변경(실례로 HDF)과 통보문넘기기(실례로 MPI)는 그우에서 실현될수 있다.

현재 체계들에서 사용되는 대부분의 병렬언어들은 Fortran이나 C의 몇개의 환경이다. 프로그람작성과 특정준위에서의 이러한 병렬언어들은 순차적인것들보다 세대확대가능성과 이질확대가능성이 훨씬 낮다. 따라서 한 병렬가동환경에서 개발된 병렬프로그람을 다

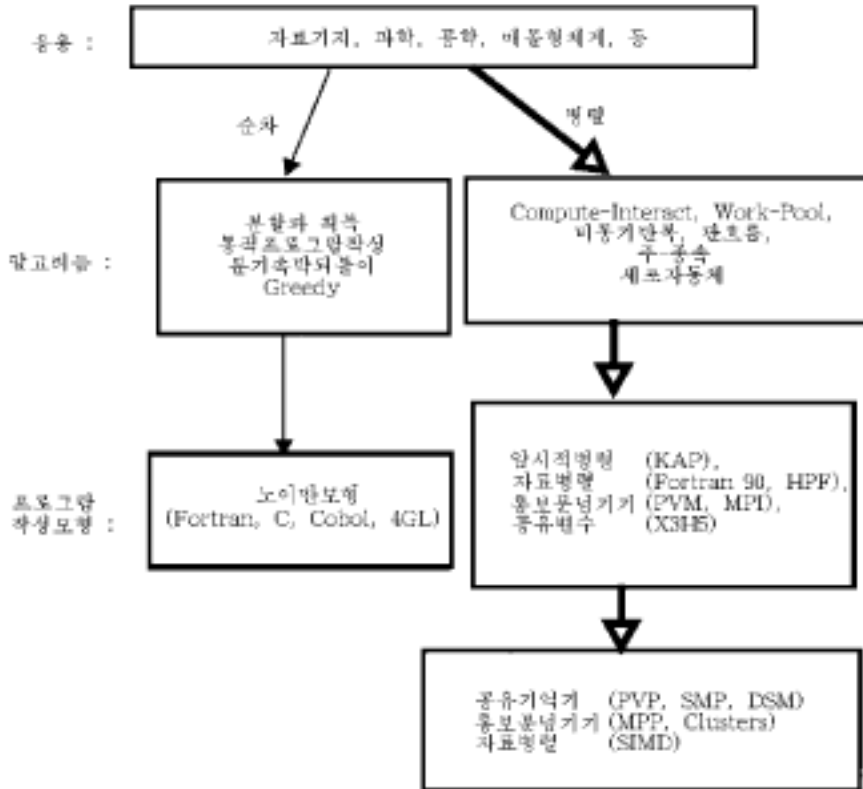



그림 2-1. 병렬프로그람작성과 순차프로그람작성의 비교

른 매개의 병렬콤퓨터들에 이식할수 있다는 가망은 적다.

병렬프로그람작성에서의 진보 앞에서와 같은 비관적인 평가에도 불구하고 지금까지 병렬프로그람작성분야에서 많은 진보가 이룩되였다.

많은 병렬알고리듬들이 개발되였다. 그 대부분이 비현실적인 PRAM모형에 토대하고 있지만 몇개는 실제적인 리용에 적용할수 있다.

현재 적은 수의 간단한 병렬알고리듬견본들이 개발되고 그것들을 받아 들이고 있다. 이 견본들에 대하여서는 12.1.1에서 고찰한다.

특정모형들이 두개의 모형에로 수렴하고 있다. 즉 PVP와 SMP, DSM에 대한 단일주소공간, 공유변수모형과 MPP와 클라스터를 위한 다중주소공간, 통보문넘기기모형에로 수렴한다. SIMD모형은 일반목적의 계산에서 없어 졌다. 그러나 그것은 여전히 특수한 목적 즉 신호, 화상, 다매체프로쎄스와 같은 매몰형응용들에서 여전히 쓸모 있다.

고수준병렬프로그람작성모형들은 세가지 표현모형 즉 자료병렬(실례로 HPF), 통보문넘기기(실례로 PVM과 MDI), 공유변수(실례로 ANSI X3H5)모형들에로 수렴하고 있다. 또한 사용자가 순차프로그람을 작성하고 콤파일러병렬성에 의하여 암시적병렬성을 끌어 내는 모형도 있다. 이 모형들은 간단하게 12.2.1에서 고찰하고 구체적으로는 4편에서 론의할것이다.

처리량 병렬쏘프트웨어개발에서 빠른 진보가 있지만 미래를 바라 볼수 있는 순차응용에 비하면 병렬응용은 매우 적다. 병렬콤퓨터체계(하드웨어와 쏘프트웨어를 다 포함하는)가 효과적으로 다중, 독립, 순차일감들의 체계처리량을 증가시키려면 순차프로그람들을 효과적으로 지원해야 하며 동시에 단일병렬응용의 응답시간을 감소시켜야 한다.

고성능병렬콤퓨터를 단일체계영상을 가지는 거대한 워크스테이션으로 보는것은 많은 사용자들이 프로쎄스강화의 우점과 다중순차일감을 실행하기 위한 기억저축가능성을 가질수 있게 한다. Gregory Pfister는 이 방식을 **직렬프로그람병렬체계(SPPS)모형**[497]이라고 부르고 있다. 역시 처리량과정으로 알려 져 있다.

2. 1. 2. 병렬프로그람작성환경

사용자의 견지에서 전형적인 병렬처리체계는 그림 2-2에서 보여 준것과 같은 구성방식를 가진다. 알고리듬은 우선 응용문제를 풀기 위해 개발된것이다. 그다음 사용자(프로그람작성자)는 그 알고리듬을 고수준언어(원천코드)로 실현할것을 요구한다. 알고리듬도, 원천코드도 명시적으로 병렬일 필요는 없다.

그다음 콤파일러는 원천코드를 어떤 병렬가동환경에서 실행하기 위한 특정코드로 변환하는데 그것은 조작체계와 기초적인 병렬콤퓨터하드웨어를 포함한다. 아쎔블러나 련결편집기와 같이 원천코드를 변환하는데 포함되는 모든 쏘프트웨어를 가진 **콤파일러**라는 용어를 리용한다. 전처리기는 원천 대 원천변환기이다. 그 실례의 하나가 널리 알려 진 C전처리기 CPP이다.

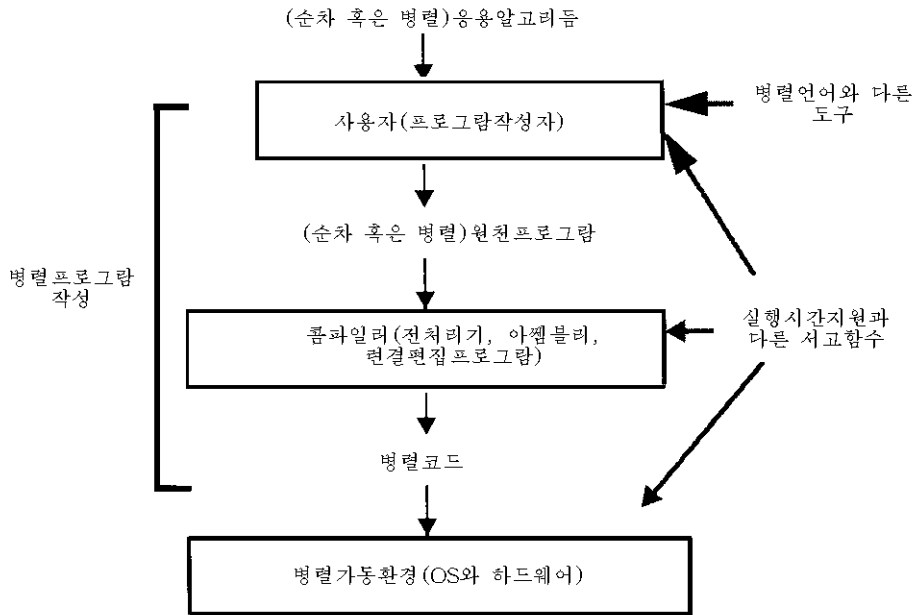



그림 2-2. 병렬처리체계의 구성요소

어떤 프로그람작성언어는 실행시간지원체계를 가지게 된다. 그것은 몇가지 필수적기

능을 사용자코드에 제공하여 적재되는 부분루틴의 모임이다. 그것들은 사용자코드가 실행을 시작할 때의 초기화, 사용자코드가 끝난 다음의 Clean up, 자료대상의 배정과 배정취소와 같은것이다. 실행시간지원부분은 서고에 의해 제공된다. 서고는 자주 사용되는 몇가지 특정규칙에 따라 콤파일된 부분프로그람의 모임인데 그것은 몇가지 필수적인 기능을 실현할 때 콤퓨터자원과의 명백하고 고수준인 대면부를 제공한다.

서고는 어떤 언어에 따르게 할수 있다. 표준 I/O를 다루는 C"stdeo"서고를 들수 있다. 서고는 또한 조작체계를 동반하는데 실례로 Solraris의 thread서고를 들수 있다. 이 서고는 다중스레드기술을 제공한다. 그것은 또한 MPI와 같이 언어독립과 가동환경독립일수 있는데 MPI는 병렬프로쎄스의 통보문넘기기모형을 지원한다.

서고부분프로그람은 사용자코드와 실행시간전에 혹은 실행시에 동적으로 사용자코드에 련결될수 있다.

환경도구 도구라는 용어는 지금까지 각이한것을 의미하는데 리용되여 왔다. 가장 넓은 의미에서 도구는 사용자응용의 개발과 실행을 돕는 어떤 하드웨어와 쏘프트웨어유틸리티를 의미한다. 모든 이런 도구의 모임은 집합적으로 프로그람작성환경(또는 간단히 환경) 으로서 취급한다.

도구의 실례들은 조작체계, 유틸리티, 프로그람작성언어들, 콤파일러들, 실행시간서고들 등이다.

환경도구들은 보통 조작체계 또는 프로그람작성환경과 관련되지 않은 도구들의 모임이다. 환경도구들은 다음과 같은 형태들을 포함한다.

- 일감관리도구들은 체계자원을 계획하고 사용자의 값을 관리하는데 사용된다. 실례들은 망대기체계(NS)와 부하공유 Fdcility(LSF)이다. 이러한 도구들은 역시 부하균형화와 검사점기능을 제공한다.
- 오유수정도구는 병렬순차응용들에서 의미적오유를 찾아 내고 그 위치를 지정하는데 사용된다.
- 성능도구들은 성능병목을 식별하도록 사용자응용을 조종하는데 사용되며 그것은 또한 성능검열수정으로 알려 져 있다.

2. 1. 3. 병렬프로그람작성방법

실제적인 병렬콤퓨터들에서 널리 사용되는 병렬프로그람작성의 모형에는 현재 네가지가 있다. 즉 암시적, 자료병렬, 통보문넘기기, 공유변수모형이다. 이 모형들은 12.2와 4편에서 론의한다. 이 병렬프로그람작성모형들은 실제적인 프로그람작성체계에서 실행되며 주로 Fortran을 확장하여 실현한다. 확장의 세가지 수단으로서 서고부분프로그람, 새로운 언어구성, 콤파일러 directive들이 있다.

- **서고부분프로그람** 순차응용에서 유효한 표준서고에 보충적으로 새로운 서고함수모임이 첨부되여 병렬성과 호상작용조작을 지원한다. 이 서고의 실례들로 MPI통

보문넘기기서고, POSix Pthreads다중스레드기능서고가 있다.

- **새로운 언어구성** 프로그람작성언어는 병렬성과 호상작용을 지원하도록 몇가지 새로운 구성에 의하여 확장된다(실례는 Fortran90에서 집합적배렬조작).

- **콤파일러지시어** 프로그람작성언어는 여전히 같지만 형식화된 설명문 즉 콤파일러지시어라고 부르는것이 보충된다.

이 방법들을 실례로 그림 2-3에 제시되였다. 모든 새로운 병렬프로그람들은 그림 2-3 ㄱ)에 있는 순차 C코드와 같은 계산을 수행한다. 서고방법은 그림 2-3 ㄴ)에서 보여

```
for ( i = 0 ; i < N ; i ++ )  A[i] = b[i] * b[i+1] ;
for ( i = 0 ; i < N ; i ++ )  c[i] = A[i]  + A[i+1] ;
```

ㄱ) 순차코드토막

```
id = my_process_id () ;
p = number_of_processes () ;
for ( i = id ; i < N ; i = i+p )  A[i] = b[i] * b[i+1] ;
barrier () ;
for ( i = id ; i < N ; i = i+p )  c[i] = A[i]  + A[i+1] ;
```

ㄴ) 서고루틴을 리용한 등가인 병렬코드

```
my_process_id(), number_of_processes (), and barrier()

A(0:N-1) = b(0:N-1) * b(1:N)
c = A(0:N-1) + A(1:N)
```

ㄷ) 배렬연산을 리용한 Fortan 90에서 등가코드

```
#pragma parallel
#pragma shared ( A, b, c )
#pragma local ( i )
{
   #pragma pfor iterate (i=0; N ; 1)
        for ( i = 0 ; i < N ; i ++ )  A[i] = b[i] * b[i+1] ;
   #pragma synchronize
   #pragma pfor iterate (i=0; N ; 1)
        for ( i = 0 ; i < N ; i ++ )  c[i] = A[i] + A[i+1] ;
}
```

ㄹ) SGI Power C에서 pragmas를 리용한 등가인 코드

그림 2-3. 3개의 병렬성방법

준다. 이것은 두개의 서고함수 my_process_id( )이다.

Number_of_processes( )는 병렬성을 리용한다. Barrier( )함수에서 모든 프로쎄스는 첫번째 순환후에 동기화되고 그리하여 두번째 순환은 첫 순환에서 갱신된 배렬 A의 정확한 값을 사용하도록 한다.

이 병렬조작은 새로운 언어구성을 사용하여 더 간단하게 할수 있다. 그림 2－.3 ㄷ)에서 례증된다. Fortran90배렬배정구성체 A(0:N-1)=b(0:N-1)*b(1:N)은 하나의 배정명령문에서 N개의 요소곱하기를 수행한다. 두 배렬배정사이에서 명시적동기화의 필요성은 없다. 왜냐하면 Fortran90명령문들은 완만하게 동기화를 진행하기때문이다. 즉 한명령문안의 모든 조작들은 다음명령문이 시작하기전에 끝난다.

콤파일러지시어방법은 그림 2-3 ㄹ)에서 보여 주었다. 병렬 pragma는 다음명령문(블로크로 발생한다.)이 병렬로 실행된다는것을 나타낸다. 공유 pragma는 세개의 배렬변수들의 병렬프로쎄스들에 의해서 공유된다는것을 나타낸다. 한편 국부 pragma는 매개 프로쎄스의 국부 *i*변수를 가진다는것을 의미한다. SGI pfor pragma장벽동기화를 발생한다.

이 세가지 방법들의 상대적인 우점과 결함은 표 2-1에 제시하였다. 서고법은 현재 가장 널리 사용된것이다. 왜냐하면 실현이 쉽기때문이다. 모든 병렬성과 호상작용가능성은 순차 C와 포트란을 보충완비하는 서고부분프로그람모임에 의하여 실현된다. 결과 새로운 콤파일러를 만들 필요는 없다. 그러나 콤파일러의 지원이 없으면 사용자는 콤파일시 해석, 오유검사, 최량화를 하지 못한다.

표 2-1 병렬프로그람작성체계의 실행을 위한 세가지 방법

방법	실례	우점	결점
서고	MPI, PVM, Cray Craft	실행하기 쉽다. 새로운 콤파일러를 요구하지 않는다.	콤파일러검사를 허용하지 않는다. 분석, 최량화
새로운 구성체	Fortran 90, Cray Craft	콤파일러검사를 허용한다. 분석, 최량화	실행하기가 어렵다. 새로운 콤파일러를 요구한다.
지령	HPF, Cray Craft	서고와 새로운 구성사이에 순차를 무시할수 있다.	

새로운 구성체방법은 서고방법과 꼭 반대이다. 새로운 콤파일러는 하나의 관리자로서 이 방법이 보다 실현하기 어렵게 한다. 그러나 새로운 콤파일러는 확장해석과 최적화해석을 수행할수 있게 하며 그것은 오유를 찾아 내고 보다 긴밀하고 효과적인 코드를 발생하는데 쓸수 있다.

콤파일러지시방법은 다른 두 방법사이의 하나의 절충으로서 볼수 있다. 그것은 하나의 보충적우점을 가진다. 즉 병렬프로그람은 바로 순차프로그람에 몇가지 형식화된 해석들을 더한것이다. 때문에 그것은 임의의 순차가동환경에서 콤파일되고 실행될수 있으며 그 과정에 콤파일러는 모든 지시어들을 무시할수 있다(그림 2-3 ㄹ)를 참고).

모든 pragma들이 무시(삭제)될 때 그것은 그림 2-3 ㄱ)의 순차 C프로그람과 같다.

프로그람작성모형을 실현하는데 세가지 방법이 사용될수 있다. 그 방법들과 프로그

람작성모형들은 모두 임의의 병렬가동환경에서 여러가지 방식으로 결합될수 있다. 실례로 Cray MPD프로그람작성모형은 세가지 방법을 모두 리용하여(새로운 구성, 서고기능, 콤파일러지시어) Cray Craft라고 부르는 집적화된 프로그람작성도구에서 자료병렬(Fortran90), 공유변수(작업공유) 그리고 통보문넘기기(PVM)프로그람작성모형을 실현한다.

2. 2. 프로쎄스, 과제, 스레드

한 병렬콤퓨터에서 사용자응용은 프로쎄스, 과제, 스레드로써 실행된다. 그것들사이의 공통성과 차이를 론의하자. 프로쎄스의 일반적인 정의는 실행되고 있는 하나의 프로그람이다. 이 간단한 정의는 분류하기 위하여 필요하다.

2. 2. 1. 추상프로쎄스의 정의

프로쎄스개념은 두가지 준위에서 고찰하는것이 쓸모가 있다. 여기서 론의될 추상적인 견지는 간단하고 병렬콤퓨터사용자들에게 적합하다.

다음절에서 프로쎄스에 대한 보다 자세한 정의를 주며 어떻게 그것이 실현되는가를 본다. 개념적으로 프로쎄스(스레드, 과제)는 네가지 구성을 가지는 하나의 동적항목으로 볼수 있다.

정의 2. 1 프로쎄스 *P*는 4개의 조 *P*=(P, C, D, S)이다. 여기서 P는 프로그람(또는 코드), C는 조종상태, D는 자료상태, S는 프로쎄스 *P*의 상태이다. 프로쎄스는 그안에 코드본문이나 자료가 없다는 점에서 동적이지만 실행중에 있다는 내용을 가지고 있다. 정확히 이것이 무엇을 의미하는가 아래에서 설명한다.

프로그람(코드) 어떤 프로쎄스가 하나의 프로그람과 관련된다. 구체적인 실례로서 다음과 같은 C코드를 고찰해 보자.

```
main(){
    Int  t=0;
    fork();fork();
    printf(   Hello!\n   ) ;
```

이 프로그람을 콤파일할수 있다(그것은 파일 hello.c안에 있다고 가정한다.). 프로그람은 $cc−0 Hi hello. c이며 그것은 간단한 명령 $Hi로써 실행한다.

조작체계는 hello라고 네번 현시하는 프로쎄스를 생성한다. 사용자에 의하여 제공되는 명확한 코드에 의하여(원천은 hello.c안에 있다.) 프로쎄스는 실행시 지원, 서고부분프로그람, 체계기능을 그림에서 보는바와 같이 사용한다. 그리고 hello.c안에 있는 원천코드를 **사용자코드**라고 부른다. 우의 프로쎄스는 printf( )를 호출하며 printf( )는 표준 C프로그람작성언어 I/O서고의 프로그람이다. 또한 fork( )도 호출하는데 그것은 Unix체계의 호출이다.

조종과 자료상태 대부분 프로그람은 명령식기계모형에 토대하며 여기서 중심개념은 상태갱신이다. 명령식프로그람은 상태기계(혹은 자동체)로 고찰될수 있고 그것은 프로그람을 초기상태로부터 하나 또는 그이상의 최종상태로 넘긴다. 우선 몇개의 용어들을 정의한다.

정의 2. 2 프로그람의 두개의 변수모임을 사용한다. 즉 자료변수들은 프로그람작성자가 자료감을 유지하도록 선포한 변수들이다. 조종변수들은 조종흐름정보를 유지하는 변수들로서 명시적으로 선포되지 않는다. 다른 말로 말하면 조종변수들은 다음에 어떤 조작들이 실행되여야 하는가와 관련한 정보들을 유지한다. 이 두 모임의 합은 프로그람변수모임을 이룬다.

정의 2. 3 임의의 시각에 프로그람의 매개 자료나 조종변수는 값에 의하여 쌍을 이루며 그것은 정의되지 않은 특수한 값이다. 시각 *t*에서 모든(자료변수, 자료값) 쌍들의 모임은 시각프로그람의 자료상태를 정의한다. 그리고 *t*시각에서 프로그람상태는 *t*에서 자료상태와 조종상태의 합이다. 때로는 *t*에서 프로그람상태를 모든 쌍(프로그람변수, 값)들의 모임이라고 말할수 있다.

정의 2. 4 프로그람은 초기상태를 가지고 출발한다. 프로그람의 원자조작(정의 1.2 참고)이 실행될 때 프로그람은 현재 상태로부터 다음상태로 변환된다. 프로그람은 원자조작을 계속 실행하면서 그것이 끝날 때까지 상태를 끊임없이 갱신한다. 그러면 프로그람은 어떤 최종상태에 도달한다.

명령식프로그람은 하나 또는 그이상의 조작의 렬들을 생성하는것으로 볼수 있으며 이 렬들은 상태기계를 초기상태로부터 최종상태로 변화시킨다. 물론 프로그람은 끝나지 않을수 있다. 프로그람이 정지하지 않는다고 보게 되는 상태에 들어 갈 때 프로그람은 발산상태에 들어 간다고 말한다.

간단히 조종변수는 프로그람계수기로서 간주할수 있다. 조종의 단일스레드를 가지는 어떤 프로세스에는 바로 하나의 조종변수 즉 프로그람계수기가 있다. 다중조종스레드를 가지는 하나의 프로쎄스에는 다중조종변수가 매개 스레드마다 하나씩 있으며 그 스레드의 프로그람계수기값을 유지한다. 다중부분프로쎄스에 대하여 전체 프로쎄스는 여러개의 조종변수를 가질수 있고 그 매개는 하나의 부분품요소프로쎄스에 대응한다.

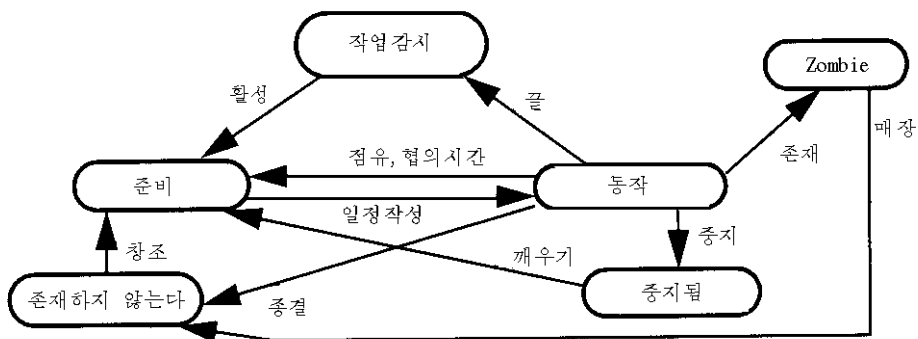



그림 2-4. 프로쎄스상태이행도표

자료변수는 사용자프로그람에서 암시적 또는 명시적으로 선포될수 있다. 실례로 Unix프로쎄스의 자료변수는 표준입력, 표준출력, 표준오유에 대하여 각각 숨은 파일지적자 stdin, stolout stdevr들을 포함하는데 그것은 명시적으로 선포되지 않을수 있다.

프로쎄스상태(상태) 프로쎄스는 어떤 시각에 일정한 상태를 가진다. 몇가지 중요한 상태들과 그것들의 이행을 그림 2−4에서 보여 준다.

- 시작할 때 프로쎄스는 존재하지 않는다(그것은 존재하지 않는 상태를 가진다.). 그것은 생성기에 의한 프로쎄스생성조작을 통하여 어미프로쎄스에 들어 간다. 새롭게 생성된 새끼프로쎄스는 실행준비되지만 실제적으로 실행(그것의 코드를 수행)을 오직 스케쥴한 다음에 시작된다. 프로쎄스가 실행하고 있는 동안에 여러가지가 일어 날수 있다.
- 그것은 실행을 계속할수 없어서 즉 페지고장이나 몇가지 다른 상황의 원인으로 정지된 상태에 들어 갈수 있다.
- 후에 다시 실행될수 있게 상황이 변화될 때 그것은 준비상태를 가지도록 다시 나타날수 있다.
- 프로쎄스를 실행하는것은 역시 높은 우선권을 가지는 다중프로쎄스에 의하여 리용될수 있거나 혹은 거기에 배당된 CPU시간토막을 소비할수 있다. 두 경우에 프로쎄스는 그자체가 실행을 계속할수 있으며 그러나 그것은 CPU자원을 주어 그 상태를 ready로 교체한다.
- 마침내 프로쎄스의 실행은 그자체가 끝날수도 있다. 그 코드가 실행을 끝낼 때 정규적으로 끝내거나 비정규적으로 끝내(중지)거나 관계 없다.

또 하나의 자주 사용되는 조작은 프로쎄스절환이다. 이것은 프로쎄스의 실행을 정지 혹은 준비된 상태로 옮기거나 실행되는 프로쎄스의 다음준비를 계획한다는것을 의미한다. 그림 2−4의 도식은 주요프로쎄스상태들을 포괄한다는것을 의미한다.

실제적인 조작체계와 병렬프로그람작성환경에서 보충적인 상태들로 하여 훨씬 더 복잡해 진다. 실례로 공유기억기병렬프로그람작성[39]에서 ANSI X3H5표준은 새로운 상태를 허락한다. 즉 프로쎄스는 실행을 끝내는데 관하여 작업감시상태에 들어 갈수 있다. 이러한 놀고 있는 프로쎄스는 어떤 새로운 작업을 할수 있도록 배정할 때 활성화된다. 이 도식은 프로쎄스의 중지로부터 초래되는 부가처리를 감소시킬수 있고 그다음 새로운 작업을 하기 위해 다시 생성된다.

우에서 준 프로쎄스개념은 사용자들 즉 병렬언어를 쓰려는 사람들에게 적합하다. 프로쎄스개념을 실현하려는 사람들, 실례로 병렬언어나 통보문넘기기서고의 제작자들에 대해서는 프로쎄스의 보다 상세한 견해가 요구된다. 프로쎄스를 실현하려면 다음의 관점 즉 실행방식, 주소공간, 문맥처리서술자, 프로쎄스조종을 고찰해야 한다.

2. 2. 2. 실행방식

실행방식에서 조작체계는 보통 다음의 구성요소들을 포함한다.

핵심부는 조작체계에서 본질적이며 필수적인 프로그람으로서 체계자원을 직접 관리하고 그밖에 프로쎄스들을 조종한다. 오직 하나의 핵심부만이 한 순간에 하나의 콤퓨터안에 있을수 있다.

쉘은 역시 명령해석기로 알려 져 있다. 그것은 조작체계에 대한 사용자대면부이다. Unix우에서 C쉘은 하나의 실례이다.

편의프로그람은 자주 사용되는 기능 즉 콤파일러, 편집기, 오유수정기 등을 제공하는 보충적조작체계이다.

콤퓨터는 프로그람을 실행하는데서 두가지 실행방식을 제공한다. 어느 방식이 적용되는가 하는것은 어떤 특정의 등록기안에 있는 하나 또는 그이상의 비트에 의하여 가리켜진다. 핵심부는 핵심부방식에서 실행하는데 감독자방식, 체계방식 또는 특권방식으로 알려져 있다.

게다가 핵심부프로쎄스들은 핵심부방식에서 실행된다. 이것들은 핵심부가 체계자원을 관리하는데 도움을 주는 핵심부에 의한 생성된 프로쎄스들이다. Unix에서 swapper프로쎄스는 하나의 실례이다.

다른 프로그람들은 사용자방식에서 실행된다. 이러한 프로쎄스를 **사용자프로쎄스**라고 부른다. 또 유틸리티(실례로 콤파일러)가 조작체계의 부분이라고 해도 그것들은 사용자프로쎄스로 실행된다는것을 알아야 한다. 사용자프로쎄스가 실현되고 있을 때 핵심부는 역시 기억에 있다. 사용자방식에서 프로그람실행은 **사용자준위프로그람**이라고 부른다. 다른것은 핵심부준위프로그람이다.

방식절환 실행방식은 사용자와 핵심부방식사이에서 뒤 또는 앞으로 절환될수 있다. 모든 콤퓨터는 핵심부방식에서 출발한다. 체계를 초기화하고 몇가지 핵심부프로쎄스를 생성한후에 핵심부는 결국 조종을 쉘로 보낸다. 그것은 보충적인 사용자프로쎄스를 생성할수 있는 하나의 사용자프로쎄스이다.

사용자프로쎄스가 실행될 때 실행방식은 다음의 두가지 상황에서 핵심부방식으로 절환될수 있다.

사용자프로쎄스가 체계호출이 핵심부로부터 봉사를 요구하게 하는것을 포함하여 동기례외가 발생될 때, 체계호출.

비동기례외(실례로 시계 또는 디스크새치기)가 존재할 때.

핵심부가 요구된 봉사를 끝낸후에 실행방식은 사용자방식으로 절환할수 있다. 핵심부에 의하여 수행된 봉사들은 다음 두 부류들로 나눌수 있다. 첫번째 부류는 어떤 특수한 사용자프로쎄스를 위하여 수행되지 않는다. 이것은 프로쎄스비동기례외를 스케쥴하고 조종하는 프로쎄스를 포함한다.

두번째 부류는 프로쎄스가 동기례외를 발생할 때 사용자프로쎄스를 위하여 수행된다.

실례로 사용자프로쎄스가 령나누기를 발생할 때 핵심부는 례외조종기를 실행하고 결국 사용자프로쎄스를 끝낸다. 또한 사용자프로쎄스가 체계호출함정(새치기)을 만들 때 핵심부는 예견된 체계봉사(실례로 I/O함수)를 실행하고 그다음 사용자프로쎄스를 계속 진행한다.

2. 2. 3. 주소공간

콤퓨터들은 어떤 대상을 가리키기 위하여 주소(또는 위치)를 사용한다. 대상은 자료이거나 코드이다. 명령에서 규정된 주소를 **가상주소**라고 부른다. 한편 처리기가 그것을 주소모선(물리)에 설정하는 주소를 **물리주소**라고 부른다.

대부분 콤퓨터체계에서 매개 주소는 물리적이든 가상이든 1byte의 정보를 가진다. 주소들은 보통 0에서 시작하여 최대까지 련속적으로 커진다.

처리기의 주소공간 어떤 처리기에 대하여 모든 가능한 가상(물리)주소들의 모임을 그 처리기의 **가상(물리)주소공간**이라고 부른다. 주소공간이란 용어는 물리적 및 가상주소공간사이를 구별하지 않는것이 중요하지 않을 때 자주 사용된다.

주소공간의 크기는 주소의 총수 A이다. 매개 주소가 한 byte를 가지므로 우리는 자주 Abyte의 **주소공간**이라고 말한다. 처리기의 가상주소공간의 크기는 명령모임구성방식에 의하여 주어 지며 물리적주소공간의 크기는 그것의 주소모선의 폭에 의하여 결정된다.

실례로 Intel Pentium처리기는 32bit 물리적주소들을 가지며 하나의 프로쎄스는 46bit 가상주소를 가진다. 프로쎄스의 견지로부터 처리기는 2^{32}=4GB의 물리적주소공간과 2^{46}=64TB의 가상주소공간을 가진다. 왜 64−TB가상주소공간이 필요한가를 다매체를 통하여 쉽게 알수 있다.

닫긴 movie서고를 만든다고 가정하자. 하나의 영화(movie)프레임은 1024×1024화소들을 가지며 매 화소는 4byte를 요구한다. 그것은 프레임당 4MB이다. 1s의 영화에는 30개의 프레임들이 있다. 압축하지 않은 두시간 영화는 2×3600×30×4MB=0.864TB를 요구한다. 언제나 4−GB의 물리주소공간이 요구되는가가 명백치 않는것은 부분적으로 물리적주소들이 오직 물리적주소공간에만 접근하기때문이다.

이 책에서 용어 기억기주소공간(간단히 기억)이 자주 주소공간대신에 사용된다. 실례로 Intel처리기는 독립적인 I/O주소공간을 가진다.

기억주소공간에서 주소는 임의의 장치를 가리키는데 사용할수 있다. 장치는 기억기, 디스크 그리고 다른 I/O장치들을 포함한다. 하나의 좋은 류사한 실례는 주소를 전화번호로 고찰하는것이다.

주기억으로 모든 물리적주소공간을 유지할 때 4GB가 큰것은 아니다. IBM PC가 나왔을 때 그것은 다만 약 0.5MB주기억을 가지고 있었다. 1997년에 하나의 IBM PC에 대하여 16 또는 32MB의 주기억을 가지는것이 보통이였다. 이 추세에 따르면 15~20년내에 개인용콤퓨터에 대하여 4GB를 내다 볼수 있을것이다. 현재의 고속콤퓨터들에서 4GB 또는 보다 큰 물리적주소공간은 분산형이든 집중형이든 공유기억기를 위해 요구된다.

실례 2. 1. Alpha 21064처리기에서 주소공간

DEC Alpha 21064처리기는 34bit 물리적주소들을 가진다. Gray는 그것을 T3D MPP를 제작하는데 리용했을 때 그것의 물리적주소공간이 너무 작다는것을 밝혔으며 매개 처리기에 주소공간을 확장하려면 새로운 소편을 보충해야 했다.

왜냐하면 Cray T3D는 2048처리기들로 확대할수 있게 설계되고 매개 처리기는 64MB

의 국부기억기를 가질수 있으며 한편 그것은 다른 처리기들과 떨어 져 있기때문이다.

분산공유기억기계면 T3D는 매개 처리기가 국부기억과 마찬가지로 원격기억기에 접근하게 한다.

그 의미는 매개 처리기가 2048*64MB=128GB 또는 37bit 물리적주소공간이 필요하다는것이다.

표 2-2는 널리 쓰이는 주소공간크기를 보여 준다. 알수 있는바와 같이 뒤에 있는 모형일수록 더 큰 공간을 가진다. 실례로 DEC Alpha구성방식은 실제적으로 가상 및 물리주소들을 위하여 64bit들을 허용한다. 오직 43bit 가상주소들이 21064처리기에서 실현되지만 그 나머지 21bit는 보존된다. 따라서 21064를 위하여 개발된 프로그람들은 그다음의 처리기우에서 실행될수 있으며 그것들은 21bit의 부분 또는 전체를 쓸수 있다.

표 2-2 일반처리기의 주소공간크기

구성방식	모형	물리주소공간크기	가상주소공간크기
Intel 80x86	8086	1 MB (2^{20} B)	1 MB (2^{20} B)
	Pentium	4 GB (2^{32} B)	64 TB (2^{46} B)
PowerPC	601	4 GB (2^{32} B)	4 PB (2^{52} B)
	620	16 EB (2^{64} B)	16 EB (2^{64} B)
DEC Alpha	21064	16 GB (2^{34} B)	8 TB (2^{43} B)
	21164	1 TB (2^{40} B)	8 TB (2^{43} B)
MIPS	R4000	64 GB (2^{36} B)	1 TB (2^{40} B)
	R10000	1 TB (2^{40} B)	16 TB (2^{44} B)

가상기억기배치(layout) 어떤 프로쎄스의 가상주소공간배치를 그림 2-5에서 간단히 보게 된다. 그것은 Unix와 같은 일반적인 조작체계에서 볼수 있다. 프로쎄스주소공간의 부분은 핵심부방식에서만 접근될수 있으며 그것을 **핵심부공간**이라고 부른다. 다른 부분은 사용자공간으로서 세토막으로 나눈다. 고정된 크기의 처리기를 가지고 있는 본문토막은 쓰기할수 없으며 다른 프로쎄스에 의하여 공유될수 있다.

표 2-2의 자료토막은 프로쎄스의 정적 및 동적자료대상들을 유지한다. 동적자료는 자료토막의 너비구역에 기억되며 그것은 실행시 늘거나 줄어 들수 있다. 탄창토막은 수속호

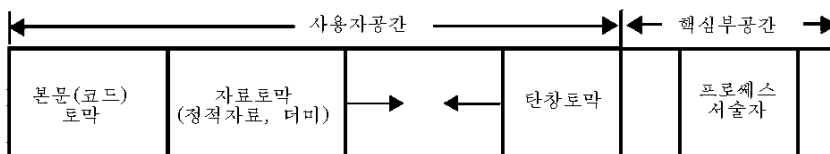



그림 2-5. 프로쎄스의 주소공간배치

출의 활성화기록을 보관하는데 사용되는데 수속호출은 다른것들가운데서 그 프로쎄스에 의하여 만들어 진것이다. 탄창 역시 실행시 늘거나 줄어 들수 있고 보통 너비와 반대방향으로 변화된다.

2. 2. 4. 프로쎄스문맥

어떤 프로쎄스문맥은 처리기등록기에 기억되는 프로그람상태의 부분이라는것이다. 모든 처리기등록기들이 프로쎄스문맥의 부분으로 되는것은 아니다. 실례로 류점등록기들이 그것들을 사용하지 않는다면 프로쎄스문맥안에는 없다.

문맥절환은 현재의 프로쎄스문맥을 보관하고 새로운 문맥을 적재하는 트랜잭션이다. 문맥절환은 방식절환이 있을 때 요구된다. 실례로 건반새치기가 사용자프로쎄스가 실행되고 있는 동안에 일어 나면 처리기는 조종을 핵심부로 넘기여 새치기조종기(새치기봉사프로그람)를 실행하게 할것이다.

그러나 처리기가 실행되기전에 사용자프로쎄스문맥을 기억기에 보관하여 등록기들을 조종기가 쓸수 있게 속박하지 말아야 한다. 새치기가 프로쎄스된후 핵심부는 사용자프로쎄스문맥을 회복하고 조종을 사용자프로쎄스가 실행을 계속하도록 뒤로 넘겨야 한다.

문맥절환은 또한 프로쎄스절환 혹은 스레드절환이 일어 날 때에도 요구된다. 프로쎄스(스레드)절환은 프로쎄스(스레드)의 실행이 상태를 준비 혹은 정지로 변화시키고 준비된 프로쎄스(스레드)가 실행되도록 스케쥴될 때 일어 난다. 스레드절환은 최소한 탄창지시기, 프로그람계수기, 상태등록기들을 변화시킬것을 요구한다. 주소공간이 변화될 때 프로쎄스절환은 모든 등록기들을 변화시킬수 있다.

2. 2. 5. 프로쎄스서술자

프로쎄스에 대한 보충적정보는 핵심부공간에서 몇가지 구성방식들로 보관된다. 가장 중요한것은 프로쎄스서술자이며 그것은 핵심부가 프로쎄스를 관리하는 정보를 포함한다. 그것들은 다음과 같다.

- **프로쎄스신임장** 그것은 프로쎄스식별자, 어미프로쎄스식별자, 사용자식별자, 그룹식별자 기타 등과 같은것이다.
- **프로쎄스상태** 준비, 실행, 정지 같은것이다.
- **문맥** 프로쎄스문맥을 유지하는 구역. 이 구역은 실례로 프로쎄스가 실행으로부터 정지로 절환될 때 문맥을 보관하는데 리용된다.
- **기억지도** 주기와 여러가지 기억기토막의 호출권한, 토막과 페지표에 대한 지시자와 같은것이다.
- **다른 프로쎄스당 정보** 열린 파일, 수신신호 등과 같은것이다.
- **대역자료구성방식** 모든 프로쎄스에 대하여 핵심부가 관리하는 대기렬과 표에 대한 지시자와 같은것이다. 실례로 모든 준비된 프로쎄스들은 준비대기렬에서 실행하기 위하여 기다리고 있다.
- **프로쎄스조종정보** 이것은 2.2.6에서 서술된다.

프로쎄스당 정보 프로쎄스서술자를 과제서술자 혹은 **프로쎄스조종블로크**라고도 부른다. 핵심부는 모든 프로쎄스에 대하여 서술자의 표나 기록을 보관한다. 프로쎄스통보들이 모든 시간에 주기억기에 있을 필요는 없다.

조작체계는 프로쎄스서술자를 두 부분으로 나눈다. 즉 프로쎄스사용자공간을 교환할수 있는 프로쎄스당 부분과 모든 시각에 주기억기에 있어야 할 다른 부분으로써 나눈다. 몇가지 실현들에서 프로쎄스당 부분은 사용자공간에 배치되지만 오직 핵심부에 의하여 접근된다.

프로쎄스당 부분의 실례는 핵심부탄창이다. 사용자프로쎄스를 대신하여 핵심부가 례외조종기(실례로 체계호출, 령나누기조종기)를 실행할 때 조종기는 수속호출하기 위해 필요하며 따라서 탄창이 필요하다. 어떤 초기실행은 바로 모든 핵심부기능들을 위하여 한개의 탄창을 리용하며 다중처리기체계에서 핵심부탄창을 병목으로 되게 한다. 많은 현대적인 실행들은 매 프로쎄스사용자공간에서 핵심부탄창과 사용자탄창을 유지하게 한다.

2. 2. 6. 프로쎄스조종

프로쎄스조종은 핵심부가 실제적으로 프로쎄스들을 관리하는 기능들을 의미한다.

프로쎄스서술자 핵심부는 프로쎄스서술자들을 만들고 정지시키며 실행하게 하고 끝내는데 사용된다. 그것은 대역자료구성방식(실례로 준비된 프로쎄스에 대한 대기렬과 프로쎄스서술자의 목록)을 관리하며 모든 프로쎄스당 자료구성방식을 관리한다.

보호 핵심부는 프로쎄스가 권한을 가지고 있는 자원들에만 접근하는것을 담보하기 위하여 폭 넓은 검사를 수행한다. 프로쎄스서술자는 여러가지 프로쎄스에 대한 특권과 권한정보를 포함하고 있다. 보통 사용자프로쎄스는 오직 사용자공간에만 접근할수 있다(그림 2-5). 그러나 핵심부공간이나 또 다른 프로쎄스의 사용자공간에 접근할수 없다. 두 프로쎄스는 Unix에서의 통보와 흐름관과 같은 특수한 프로쎄스간 통신(IPC)을 통하여 통신한다. 어떤 체계들은 공유주소공간을 통하여 프로쎄스들을 통신할수 있게 공유기억기구역을 허용한다. 어떤 콤퓨터체계에서 모든 기억기와 I/O장치들은 주소로서 참조된다는것을 알아야 한다. 주소공간을 보호하는것으로서 핵심부는 모든 기억기, I/O장치들, 프로그람과 자료파일 등에 대한 보호를 확보한다.

일정작성기 이것은 자원(처리기, 기억기와 I/O)을 일정작성기라고 부르는 핵심부프로그람에 의하여 프로쎄스들에 배정하는것을 의미한다. 일정작성기는 공평해야 하며 효과적이여야 한다. 프로쎄스서술자는 프로쎄스우선권정보를 포함한다. 일정작성기는 같은 우선권을 가지는 프로쎄스들이 체계자원에 대한 같은 접근를 가지며 높은 우선권프로쎄스일수록 더 좋은 접근을 가지게 한다면 공평하다. 일정작성기의 중요한 형태들은 그림 2-6에 실례로 보여 주었다.

실례로 어떤 프로쎄스는 부족되지 말아야 한다. 바꾸어 말하면 어떤 속박되지 않은 처리기를 실행하기 위하여 언제나 기다리게 하여야 한다. 일정작성기는 낮은 부가처리를

일으키면 효과적이다. 가장 간단한 체계는 오직 단일사용자프로쎄스가 한 시각에 활성화되게 하는것이다. 다시 말하면 한명의 사용자만이 한 시각에 체계를 리용할수 있게 한다. 사용자프로쎄스가 I/O기능을 완결하기 위하여 기다리고 있을 때 다른 프로쎄스들은 속박되지 말아야 처리기자원을 사용할수 있다.

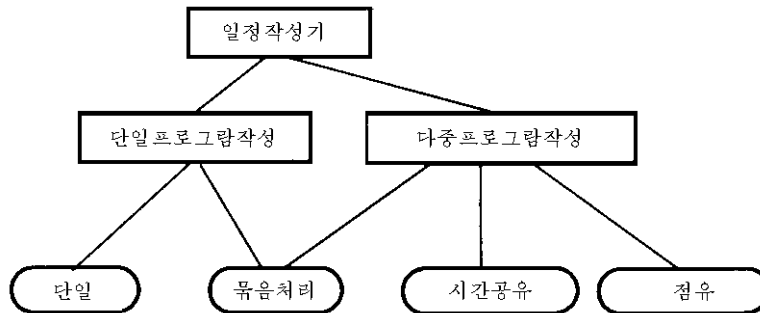



그림 2-6. 핵심부에서 여러가지 형태의 일정작성기

이 자원들은 단일사용자에게 전용으로 제공된다. 이러한 체계를 단일사용자 단일-과제기능(유일프로그람기능)체계라고 부른다. Microsoft DOS는 하나의 실례이다. 보다 정교한 체계들은 단일다중처리기들의 자원을 동시에 리용하게 한다.

이러한 단일사용자다중과제체계의 한 실례는 Microsoft Windows이다. Unix와 Windows NT는 다중사용자다중과제처리(또는 다중프로그람처리)체계의 실례들이며 그것은 많은 사용자들이 체계자원을 동시에 공유하게 한다.

자원공유는 다음과 같은 여러가지 형식을 가진다.

- 전용제공방식에서 사용자프로그람은 요구되는 자원을 배타적으로 사용한다. 그것은 프로그람이 끝날 때까지 자원을 주지 않는다. 다른 말로 말하면 자원은 공유되지 않는다.
- 묶음프로쎄스방식에서 사용자프로쎄스는 일단 동작하도록 일정작성되면 완성될 때까지 실행한다. 그러나 다중프로그람작성 묶음프로쎄스체계에서 프로쎄스는 I/O조작의 완성과 같은 사건을 위하여 기다리고 있을 때 처리기자원을 자발적으로 준다.
- 시간공유방식에서 여러 사용자프로쎄스는 동시에 한 처리기우에서 호상 배치되는 형태로 실행된다. 처리기시간은 짧은 시간토막(또는 량자)으로 분할된다. 매개 프로쎄스가 한토막동안에 실행되면 그다음 처리기를 다른 프로쎄스에 양보한다.
- 독점체계에서 높은 우선권을 가지는 프로쎄스는 처리기를 낮은 우선권을 가지는 프로쎄스로부터 떼낼수 있고 지어 이때 후자는 현재의 시간토막을 다 사용하지 못한다.

실례 2.2. 시간공유 대 묶음프로쎄스

시간공유기술이 특히 호상작용응용들에서 쓸모 있으며 그것은 말단으로부터의 I/O조

작을 요구하고 빠른 응답시간을 가진다. 10개의 프로쎄스가 하나의 처리기를 쓰고 있다고 하자. 하나의 호상작용프로쎄스의 프로쎄스시간은 5s가 요구된다. 다른 프로쎄스들은 모두 1000s가 요구된다.

묶음프로쎄스일정작성기가 호상작용프로쎄스를 다른 아홉개 프로쎄스다음에 일정작성기하였다면 묶음프로쎄스체계에서 사용자는 호상작용프로쎄스 최종결과를 얻도록 허락한 다음에는 900s동안 기다리게 된다.

그러나 시간공유체계에서 사용자는 오직 50s동안 기다리게 된다. 이 실례의 자세한 것은 련습으로 남겨 둔다.

실례 2.3 실시간체계에서 독점적일정작성기

독점적일정작성기는 실시간체계에서 가치가 있다. 그것은 일정한 체계들이 일정한 시간제한안에서 끝날것을 요구한다. 실시간프로쎄스가 5s의 처리기시간이 필요하며 10s내에 끝날것을 요구한다고 하자. 실례 2.2에서 본바와 같이 이 요구는 간단한 시간공유에 따르면 9개이상의 프로쎄스가 존재할 때 만족될수 없다. 그러나 독점적인 일정작성기를 가지면 실시간프로쎄스는 다른 프로쎄스들을 독점할수 있고 5s내에 끝날수 있는데 이 동안에 부가처리는 무시된다.

실례 2.4. 다중프로그람화된 묶음프로쎄스

병렬응용이 두 처리기다중콤퓨터에서 묶음프로쎄스방식으로 실행된다고 가정하자. 그러면 실행시간은 응용이 전용제공방식으로 실행될 때 보다 4배나 더 길게 관측된다.

왜 그런가?

하나의 답변은 묶음방식은 자원이 전용으로 제공되여 사용한다는것을 의미하지 않는다는것이다. 처리기가 체계호출을 통하여 통신조작을 수행할 때 그것은 처리기를 또 다른 처리기에 주며 그 처리기는 긴 처리기시간을 요구하게 된다. 이것은 그림 2-7의 실례를 통하여 보게 된다.

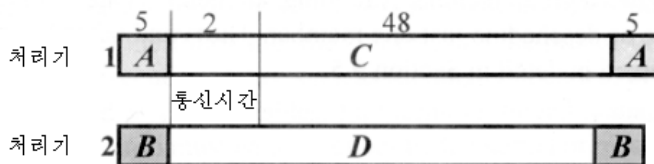



그림 2-7. 다중프로그람화된 묶음프로쎄스

병렬응용이 두개의 프로쎄스 A와 B를 가지고 매개 처리기에 배정되고 계산에 10s를 요구한다. 역시 2s의 통신부가처리가 있다. 전용제공방식에서 프로그람의 두 처리기들에 대해서 전체 12s 걸린다. 처리기들은 통신이 진행되는 동안 기다리게 된다.

다중프로그람화된 묶음프로쎄스방식에서 기다리고 있는 처리기는 통신을 계산과 중첩시켜 편리하게 할수 있다. 프로쎄스 A와 B가 통신을 호출할 때 그것들은 처리기들을 프로쎄스 C와 D에 양보한다. 그러나 A와 B는 C와 D가 그것들의 계산을 끝내고 처리기들을 양보할 때 처리기에 되돌아 올수 없다. 경과시간(혹은 벽시계시간)은 60s로 증가한다.

병렬콤퓨터들에서 다음의 용어들은 지금까지 일정작성기를 위하여 사용되여 왔다. 공간공유는 하나의 프로쎄스를 한 처리기에 배정하는것을 의미하며 서로 다른 응용들은 서로 다른 처리기모임(자주 분할부분으로 분리운다.)에 배정된다. 다중프로그람작성방식을 가리키는데 그 방식에서 서로 다른 응용의 다중프로쎄스들은 같은 프로쎄스에 7배정될수 있다. 이 프로쎄스들은 그림 2-3에 있는 묶음, 시간공유(시간토막화) 또는 독점방식에 의하여 일정작성기된다.

병렬콤퓨터들의 일정작성기는 병렬응용이 호상작용프로쎄스들을 가지는가를 반드시 고려해야 한다. 이 프로쎄스들을 개별적인것으로 일정작성기하면 독립적인 프로쎄스들은 현저히 성능을 낮춘다.

실례 2.5. 병렬처리기의 일정작성기에서 최악의 경우

그림 2-6을 다시 고찰하자. 통신이 동기화된다고 가정하자. 바꾸어 말하면 두 프로쎄스 A와 B가 함께 실행되여야 한다고 가정하자. 프로쎄스 A가 통신코드실행을 시작할 때 B는 더이상 실행되지 않는다는것(실례로 시간초과 또는 먼저 차지)을 볼수 있다. 벌써 더 나쁘게 B가 바뀌여 진다. 따라서 A는 B가 다시 실행될 때까지 오래동안 기다려야 한다.

그러나 이 시각에 A는 그것의 처리기시간토막을 다 리용하였으므로 시간이 초과된다. 따라서 B는 기다려야 한다. 이 파다상태는 오래동안 계속될수 있다. 이 파다상태문제를 해결하기 위한 기술은 그룹일정작성기이다. 그 기본사상은 호상작용하는 처리기들은 하나의 항목(그룹)으로서 일정작성기되여야 한다는것이다. 그룹성원이 실행하고 있을 때 모든 성원들은 실행하고 있다. 꼭 하나의 성원이 일정작성기할수 없거나 바로 하나의 성원만을 일정작성하게 한다. 이것들과 다른 일감일정작성기문제들은 9.5에서 구체적으로 론의된다.

2. 2. 7. 프로쎄스의 변종들

일반적인 조작체계프로쎄스들은 개별적인 주소공간을 가진다. 이 개념은 다중사용자, 다중과제환경에서 안전과 보호에 의한 의미를 가진다. 모든 사용자와 체계프로쎄스들이 같은 주소공간에 있을 때 대혼란이 초래될수 있다. 그러나 이 개별적주소공간개념은 프로쎄스관리가 시간을 소비하게 한다. 실례로 Unix프로쎄스가 fork( )체계호출을 실행하여 새끼프로쎄스를 생성할 때 그 프로쎄스에 대하여 새로운 주소공간이 생성되여야 한다. 이것은 기억이 배치되여야 한다는것, 자료토막과 어미에 대한 서술자가 복사되여야 한다는것, 실행시탄창이 새끼를 위해서 설정되여야 한다는것을 의미한다. 그러면 성능상 리유로 하여 Unix프로쎄스들을 무거운 프로쎄스라고 말한다. 프로쎄스생성과 절환의 높은 부가처리는 병렬프로쎄스에서 하나의 불리한 결과를 가질수 있다. 3장에서 볼수 있지만 하나의 Unix처리를 생성하는데 수십만의 CPU주기가 걸리며 다량의 자료모임을 가지는 실제프로쎄스가 분산형기억콤퓨터우에서 생성될 때 훨씬 더 많다. 따라서 무거운 병렬프로쎄스들이 조잡한 계산들을 가지지 않는 한 중부하병렬프로쎄스는 확대가능병렬콤퓨터에서 적합치 않다. 병렬성을 리용하려면 가벼운 프로쎄스들을 진행하여야 한다.

가벼운 프로쎄스의 개념(**스레드**라고도 부른다.)은 몇개의 조작체계, 스레드서고, 병렬

프로그람작성언어들에서 제기되고 실현되였다. OS프로쎄스와 스레드의 주요차이는 무거운 OS프로쎄스안에서 다중스레드들은 처리기와 같은 주소공간(프로쎄스서술자를 포함하면서)을 공유하며 존재할수 있다는것이다. 하나의 프로쎄스가 생성될 때 그것은 보통 기초스레드라고 부르는 단일스레드를 가진다. 임의의 스레드는 보통적인 스레드들을 스레드생성조작에 의하여 생성할수 있는데 그것은 아래서 보는바와 같은 전형적인 형식을 가진다.

Thread_cread(foo,argment1,⋯,argmentn):

이러한 조작은 프로쎄스호출과 대단히 류사하다. 그것은 스레드를 생성하여 n인수를 가지는 함수 foo를 실행한다. 개별적인 실행시 탄창은 새로운 새끼프로쎄스에 대하여 설정될것을 요구하는데 그것은 나머지주소공간 (코드토막, 자료토막, 프로쎄스서술자)을 어미스레드와 공유한다.

기억배정이 훨씬 적고 복사가 요구되기때문에 스레드를 만드는것은 무거운 프로쎄스를 생성하는것보다 훨씬 더 빠르다. 사용자준위스레드는 핵심부를 동반하지 않고 생성될수 있는데 그것은 방식스위치부가처리를 절약한다.

어떤 스레드는 스레드단위의 탄창보다 더 많은것을 요구한다. 다른 정보들은 보관되여야 하는데 그 정보는 다음과 같은것들이다. 문맥 즉 프로그람계수기를 포함한 처리기등록기, 실행, 중지, 준비와 같은 실행상태 그리고 스레드비공개변수를 유지하는 스레드단위자료구역이다. 게다가 스레드는 핵심부 혹은 사용자준위스레드서고에 의하여 관리(생성, 끝내기, 일정작성기 등등)되여야 한다.

이 책에서 용어처리는 문맥이 명백하거나 무게와 관련이 없을 때에 무거운 프로쎄스 또는 스레드를 가리키도록 사용한다. 한편 우리는 다음의 약속을 규정한다.

Task=process=heavy_weighted process=OS process
Light_weighted process=thread

2. 3. 병렬성문제

병렬프로그람작성은 순차프로그람작성보다 더 복잡하며 많은 문제들이 더 생긴다. 그러므로 사용자프로그람에서 병렬성을 규정하는데로부터 발생하는 문제들을 론의한다.

2. 3. 1. 프로쎄스에서의 동질성

이것은 병렬프로그람에서 요소프로쎄스들사이의 류사성을 의미한다. 다음과 같은 세개의 기본적인 가능성이 있다.

- **SPMD** : 단일프로그람다중자료(SPMD)프로그람에서 요소프로쎄스들은 같은 코드가 서로 다른 자료령역에서 다중프로쎄스에 의하여 실행된다는 점에서 동질이다.

- **MPMD** : 다중프로그람다중자료(MPMD)프로그람의 요소프로쎄스들은 다중프로쎄스를 서로 다른 코드로 실행한다는 점에서 동질이다.
- **SIMD** : SPMD와 MPMD프로그람들은 둘 다 각이한 명령들이 같은 시각에 서로 다른 프로쎄스에 의하여 실행될수 있다는 점에서 MIMD이다. SIMD프로그람은 SPMD보다 제한적이며 여기서 다중프로쎄스의 같은 코드가 실행해야 할뿐아니라 역시 그것들은 모두 같은 시각에 같은 기계명령을 실행해야 한다. 다른 말로 SIMD프로그람은 SPMD프로그람의 특수경우이다.

이 책에서는 MIMD프로그람들에 초점을 두고 고찰하였다. 그것은 실제응용들에서 우월한 병렬프로그람들이며 대부분병렬콤퓨터들이 MIMD기계이다. MIMD프로그람들은 보통 병렬블로크구성이다. 다중코드방법에 의하여 규정된다. 한편 SPMD프로그람들은 보통 병렬순환구성, 자료병렬구성 또는 아래서 론의되는 단일코드방법에 의해서 규정된다.

두가지 보충적인 용어들이 책에서 자주 제기된다. 자료병렬프로그람은 일반적으로 SPMD프로그람 특히 오직 자료병렬구성(Fortran90에서와 같이)사용자프로그람을 가리킨다. 그것은 자료령역이나 자료구성방식에서 병렬성을 리용하는것을 강하게 요구한다. 기능병렬(단자 또는 과제병렬이나 조종병렬로 알려 져 있다.)프로그람은 보통 MPMD프로그람과 동의어로 쓴다. MDMD(기능-병렬)와 SPMD(자료병렬)양상은 하나의 병렬프로그람에서 혼합될수 있다.

병렬블로크MDMD프로그람을 표현하는 방법은 parbegin과 parend구성을 사용하는것이다. 이 구성방식화된 구성들은 원래 Dijkatra에 의하여 제안되였으며 cobegin과 wend로 알려 져 있다. 그것들은 언제나 다음과 같이 쌍으로 사용된다.

Parbegin S_1 S・…S_n parend

는 병렬블로크로 불리우며 여기서 S_1S_2・…S_n은 부분품프로쎄스이다. 그것은 각이한 코드를 포함한다. 병렬블로크는 일반적인 순차언어들에서 볼수 있는 순차블로크 즉 begin S_1 S・…S_n에 대비하여 그렇게 부르게 되였다.

병렬블로크가 실행될 때 병렬블로크의 n개 요소프로쎄스 S_1, S_2, …, S_n은 동시에 실행출발한다. 그것들은 서로 독립으로 실행하며 매개는 자기 단계에서 실행한다. 특수한 호상작용조작들만 실행되여 이 프로쎄스들을 조화시킬것을 요구한다. 병렬블로크는 모든 n부분프로쎄스들이 끝날 때 끝난다.

병렬순환 하나의 병렬블로크안에 있는 모든 프로쎄스들이 같은 코드를 공유할 때 병렬블로크는 병렬순환이라고 부르는 간략표기법으로 다음과 같이 표시할수 있다.

Parbegin Process(1)…Process(n)parend

이것은 다음의 병렬순환으로 간단화할수 있다.

Parfor(i:=1;i<=n;i++){Process(i)}

병렬순환은 자주 SPMD병렬프로그람들을 서술하는데서 사용된다.

조작체계 또는 망체계와 같은 체계프로그람의 프로쎄스들은 보통 이질적이다. 많은 병렬계산프로그람들은 특히 대량적인 병렬콤퓨터를 위한것들은 고도로 이질적이다. 1000−처리기콤퓨터에 대하여 완전히 이질적인 병렬프로그람을 쓰는것은 어렵다. 왜냐하면 그것은 매개 처리기에 대하여 서로 다른 프로그람을 써야 하기때문이다.

병렬프로그람작성환경에서 동질 및 이질적인 프로쎄스에 대하여 제공하는것은 기대할만하다. 그러나 SPMD를 지원하는것은 주로 확대가능콤퓨터들에 대해서는 충분하다. 각이한 코드들의 수가 작을 때 SPMD프로그람을 써서 MPMD를 꾸며 낼수 있다. 실례로 MDMD코드 parbegin A; B; C; parend는 SPMD병렬순환과 등가로 표현할수 있다.

```
Parfor(i=0; i<3; i++){
    If (i=0) A;
    If (i=1) B;
    If (i=2) C;
}
```

다중코드와 단일코드. 현재 병렬콤퓨터 특히 MPP와 COW에서 사용되는 많은 프로그람작성언어들은 병렬블로크나 병렬순환구성을 제공하지 않는다. 이러한 체계들에서 MPM병렬성을 다중코드방법에 의하여 서술한다. 실례로 parbegin A; B; C; parend를 서술하는데서 사용자는 세가지 프로그람을 쓰고 그것을 콤파일하여 세가지 실행프로그람 A, B, C를 발생시킬 필요가 있다. 실행프로그람들은 셸스크립트에 의하여 세개의 프로쎄스마디들에 적재된다.

```
Run Aon node 1
Run Bon node 2
Run Con node 3
```

프로그람 A, B, C는 순차프로그람 더하기반복을 위한 서고호출이다. SPMD프로그람은 단일코드방법을 사용하여 서술될수 있다. 실례로 병렬순환 parfor(i:=0; i<N; i++){foo(i)}를 서술하는데서 사용자는 오직 다음과 같이 하나의 프로그람을 쓸 필요가 있다.

```
Pid=my_process_id();
Humproc=number_of_processes();
For(i=pid;i<n;i=i+numpro) foo(i);
```

이 프로그람은 하나의 실행프로그람 A로 콤파일러된다. 그것은 명령 run A_numnodes을 실행하여 n마디들에 적재된다. 단일코드방법은 사용자에게 더 편리하다. 그것은 같은 프로그람이 SPMD방식에서 각이한 수의 마디들에서 반복하여 실행되게 한다는것을 알아야 한다.

자료-병렬구성 자료병렬언어들에서(실례로 Fortran 90과 MPF) SPMD병렬성은 자료병렬구성을 사용하여 구성될수 있다. 실례로 병렬순환

```
parfor(i=1; i<=N; i++){C[i]=A[i]+B[i];}
```

를 서술하기 위하여 사용자는 배렬배정 C=A+B 또는 다음순환

```
forall(i=1, N)   C[i]=A[i]+B[i]
```

를 쓸수 있다.

2. 3. 2. 정적병렬성과 동적병렬성

프로그람의 구성방식은 그것이 자기의 부분품요소들로 이루어 지는 방도를 가리킨다. 실례로, 코드 if(c) S1 else S2은 구성방식 "if(…)…else…"를 가지며 세개의 부분품 C, S1, S2를 가진다. 프로그람은 그것의 구성방식과 요소프로쎄스들이 실행시간에 앞서(실례로 콤파일러, 련결편집시, 또는 적재시) 결정될수 있을 때 정적병렬성을 표시한다. 다른 경우에 프로그람은 동적병렬성을 표시한다고 말하며 그것은 프로쎄스들이 실행시에 생성되고 끝낼수 있다는것을 의미한다.

정적병렬프로그람보다 적은 실행시간부가처리를 가지므로 동적병렬프로그람들보다 더 효과적으로 되게 하는 경향성이 있다.

왜냐하면 많은 초기화조작(기억배정, 탄창배정, 체계표초기화 등등)을 피할수 있기때문이다. 동적병렬프로그람은 보다 유연해 지는 경향을 가진다. 정적병렬성은 2.3.1의 구성들에 의하여 표현될수 있다(실례로 병렬블로크, 병렬순화). 동적병렬성은 보통 몇가지 종류의 fork와 join조작을 통해서 표현된다. 그것들은 또한 단일코드나 다중코드방법을 사용하여 서술될수 있다. 실례로 병렬블로크 parbegin P, Q, R, parend는 P, Q, R가 있으면 정적이다. 그러나 코드 while (c>0) begin fork (foo(c)); c:=boo (c); end는 동적병렬성을 리용한다. 왜냐하면 그것은 다만 실행시에 fork조작이 새로운 프로쎄스를 생성하도록 몇번 실행되는가 하는것으로 알려 져 있기때문이다.

Fork/Join 지금까지 fork/join의 여러가지 견본들이 있었다. 우리는 한가지 실례를 사용하여 일반적내용을 설명한다.

다음프로그람은 세개의 프로쎄스를 가지는데 여기서 A는 주프로쎄스로서 프로그람이 실행을 시작할 때 자동적으로 생성된다. 즉

```
Process  A:        Process  B:          Process  C:
begin              begin                begin
Z:=1;              fork(C);             Y:=boo(Z);
fork(B);           X:=foo(Z);           end
T:=foo(3);         join(C);
end                output(X+Y);
                   end
```

프로쎄스 A가 fork조작을 실행하여 새로운 프로쎄스 B를 생성하자. 여기서는 A와 병렬로 실행을 시작한다. 다른 말로 말하면 B가 실행되는 동안에 A는 그것의 그다음명령문[바꾸어 말하면 배정 T:=foo(3)]을 실행하는데로 넘어 간다.

A를 B의 **어미프로쎄스**라고 하며 B를 A의 **새끼프로쎄스**라고 말한다. 두 프로쎄스는 비동기로 실행된다. B가 실행되기전에 A를 끝내는것은 가능하다(바꾸어 말하면 그것은 end명령에 도달한다.). 어떤 경우에 A는 B가 끝날 때 끝날수 있다. 일반적으로 어미프로쎄스는 모든 새끼들이 끝나기 전에는 끝날수 없다.

때때로 어미는 그것의 코드끝에 도달하기전에 어떤 새끼를 기다리게 한다. 그것은 Join명령문을 사용하여 수행될수 있다. 우의 코드들에서 프로쎄스 B는 련이어 또 다른 프로쎄스 C를 생성한다. B에 있는 출력명령문이 C에 의하여 계산되는 변수 Y의 값을 요구하기때문에 join(c)명령문은 출력명령문앞에 삽입된다. Join(c)명령문은 B가 임의의 다음 명령문실행에로 넘어 가기전에 C가 끝날 때까지 기다리게 한다.

Fork와 join은 대단히 융통성 있는 구성들이다. 그러나 그것들은 구성방식화되여 있지 않다. 그것들은 순차언어들의 goto와 류사하며 심중하게 사용되여야 한다. 많은 병렬계산프로그람들은 오직 병렬블로크와 병렬순환만을 사용하여 서술될수 있다.

2. 3. 3. 프로쎄스그룹화

병렬응용에서 프로쎄스들은 보통 독립이 아니다. 그것들은 서로서로 호상작용할 필요가 있다. 이미 2.2.6에서 호상작용하는 프로쎄스들을 그룹으로 일정작성기하는것이 필요하다는것을 보았다. 2.4에서 그룹개념은 집합적호상작용을 지원하는것과 같이 다르게 사용한다는것을 고찰하였다.

프로쎄스그룹은 프로쎄스의 순서화된 모임이다. 성원프로쎄스의 수를 **그룹크기**라고 부른다. 매 그룹은 그룹 ID를 가진다. 그것은 병렬프로그람에서 그룹을 유일하게 식별한다. 매 성원프로쎄스는 그룹에서 위수를 가진다. 그것은 보통 0부터 $n-1$까지의 옹근수이고 n은 그룹크기이다. 성원프로쎄스는 위수와 그룹 ID에 의하여 유일하게 식별될수 있다.

그룹개념을 지원하는데서 병렬프로그람언어는 그룹을 관리하는 기능들을 제공할 필요가 있다. 그것은 그룹의 생성, 소거, 그룹 ID, 그룹성원 및 성원위수 등등을 알아 보기와 같은것이다.

2. 3. 4. 배 정 문 제

병렬프로그람은 일정한 자료대상에 대하여 일정한 계산(작업부하)을 수행해야 한다. 배정은 자료의 작업부하를 프로쎄스들로 구역분할하는것과 프로쎄스들을 마디들(처리기들)에로 넘기는것을 의미한다.

항목 1.3.3과 식 (1.4)에서 좋은 배치도식은 작업부하(바꾸어 말하면 δ/w를 감소한다.)를 균등하게 하며 부가처리(바꾸어 말하면 $(t_p+t_0)/(wt_f)+(\alpha t_0)/t_f$)를 최소화하여 체계가 휴식하거나 호상작용하면서 시간을 랑비하는 대신에 대부분이 지적된 시간동안에 계산하면서 일하게 해야 한다. 이것은 병렬성을 잃지 말고 수행되여야 한다는것이다.

병렬성 구역분할의 중요한 과제는 적당한 병렬성과 립도를 선택하는것이다. 병렬프로그람의 병렬성정도(DOP)는 동시에 실행될수 있는 요소프로쎄스의 수로서 보통 정의된다. 량적척도로서 평균병렬성으로 알려 져 있는것이 있는데 3.5.1에서 정의된다. 그것은 자주 프로그람의 전체 DOP를 평가하는데 사용된다.

립도 립도는 두개의 병렬성사이에 실행되는 계산작업부하 또는 호상작용으로 정의된다. 위상병렬모형(1.3.3)에 대하여 거침도는 한개의 처리기가 초결음에서 수행하는 계산작업부하 *W*로 정의할수 있다.

립도의 단위는 명령의 수, 류점조작의 수 또는 초이다. grain과 크기는 흔히 작은, 중간, 큰것으로서 정의된다. 대략적인 분류는 결과크기가 200보다 작으면 된다. 200부터 2000까지는 중간, 수천 또는 그 이상 의 계산조작으로서는 크다고 본다.

립도란 말은 때때로 프로쎄스크기 바꾸어 말하면 어떤 부분품프로쎄스(첫 단계안의 것이 아니다.)안에 있는 프로쎄스크기를 가리키는데 사용하군 한다. 이런 식으로 조작준위(또는 명령준위)병렬성은 한 병렬프로그람안에 있는 부분품프로쎄스가 하나 또는 몇개의 계산조작 또는 기계명령으로 작업할 때 사용한다. 용어 블로크준위는 개별적프로쎄스의 크기가 배정명령문들로 이루어 진 블로크인 경우를 의미한다. 블로크준위병렬성의 특수한 경우는 순환준위인데 여기서 순환의 몇개의 프로쎄스를 생성한다. 매개는 몇개의 배정병렬로 이루어 지는 반복을 실행한다. 부분품프로쎄스의 매개가 수속호출로 구성될 때 수속준위병렬성을 가진다. 때때로 그것을 **과제준위병렬성**이라고 부른다. 병렬성정도와 grain크기는 자주 상반되며 거기서 다른것이 같다고 하면 grain크기의 증가는 병렬성을 감소시키는 경향이 있으며 grain크기를 감소시키는것은 병렬성정도를 증가시킨다. 한편 실제적인 병렬콤퓨터들에서 병렬성정도와 통신부가처리 그리고 동기화는 비례관계를 가진다. 다른것이 같다고 하면 병렬성정도의 증가는 흔히 부가처리를 증가시키고 병렬성정도의 감소는 부가처리를 감소시키는 경향이 있다.

암시적배정과 명시적배정 명시적배정에서 사용자는 자료와 작업부하를 어떻게 배정하겠는가를 명시적으로 서술할 필요가 있다. 암시적배정에서 이 과제는 콤파일러와 실행시 체계에 의하여 수행된다. 다양한 결합형태들이 가능하다. 실례로 대칭다중처리기들에서 보편적인 방법은 프로쎄스들이 접근할수 있게 자료들이 중심공유기억안에 놓이게 하는것이다. 작업부하는 프로쎄스들에 정적으로 또는 동적으로 분산된다. 한 프로쎄스가 한 토막의 작업부하에 배정될 때 그것은 공유기억기로부터 필요한 자료를 받는다.

분산형기억체계들에서 널리 사용되는 방법은 소유자계산규칙이다. 즉 자료는 우선 프로쎄스들안에서 분산된다. 프로쎄스 P가 변수 *x*에 배정될 때 P는 *x*의 소유자라고 부른다. X와 관련한 계산은 소유자프로쎄스에 의하여 수행된다. 자료와 작업부하배정은 성능에 큰 영향을 준다. 중요한 연구과제는 대부분시간동안에 프로쎄스가 요구하는 자료들이 가까이 놓이도록 어떻게 자료들을 배정하겠는가 하는것이다. 이것을 자료국부성의 리용이라고 부른다. 그것은 캐쉬, 기억기오유, 통신부가처리를 줄인다.

2. 4. 호상작용과 통신문제

호상작용은 이것들의 프로쎄스가 서로의 트랜잭션에 영향을 주는 활동이나 조작이다. 어떤 사람들은 간단히 호상작용조작을 **통신**이라고 부른다. 우리는 용어 호상작용을 쓴다. 왜냐하면 그것에는 더 많은 의미가 담겨 져 있기때문이다. 호상작용과 관련한 몇가지 중요한 문제들이 있다. 즉 병렬체계에 지원되는데 필요한 호상작용과 호상작용방식, 형태, 경쟁적인 호상작용과 협동호상작용들이다.

2. 4. 1. 호상작용조작

간단히 세가지 조작 즉 자주 사용되는 호상작용의 형태들을 소개한다. 즉 통신, 동기화, 집합조작들이다. 이 책에서 이 조작들은 주로 종합적으로 **통신**이라고 부른다. 그러나 그것들사이를 구별하는것이 중요한다. 왜냐하면 그것들은 구성방식과 프로그람작성지원에서 요구가 서로 다르기때문이다.

통신 통신조작은 자료값들을 둘 또는 그이상 프로쎄스들에서 교환한다. 공유기억기프로그람에서 한 프로쎄스는 공유변수값을 계산할수 있고 공유기억기안에 그것을 기억할수 있다. 다음에 또 다른 프로쎄스는 이 값을 변수를 참조하여 받을수 있다. 이것을 공유변수를 통한 **통신**이라고 부른다.

수속준위병렬성을 사용하는 다중처리기프로그람은 역시 어미프로쎄스가 어떤 수속을 분기시켜 실례로 fork(foo(x))를 실행하여 새끼프로쎄스를 생성하게 할수 있다.

자료값들은 새끼프로쎄스와 어미프로쎄스사이에서 파라메터로써 교환될수 있다. 이것을 파라메터교환을 통한 **통신**이라고 부른다. 마지막으로 다중콤퓨터모형에서 프로쎄스들은 통보문넘기기를 통하여 통신할수 있다.

동기화 동기화조작은 프로쎄스들이 서로 다른것을 기다리게 하거나 기다리는 프로쎄스들이 실행을 계속하게 한다. 동기화조작에는 각이한 형태들이 있다.

- **원자성** 프로쎄스가 주로 하나의 단일한 원자조작으로서 조작렬을 요구한다(정의 1.2 참고). 아래에서 실례를 보자.

```
Parfor(i=1;   i<n;   i++){
      Atomic{x=x+1;   y=y−1;}
}
```

실마리어 atomic는 *n*프로쎄스중에서 매개가 두개의 배정을 하나의 원자조작으로 실행해야 한다는것을 가리킨다. 병렬체계에서 원자성을 실행하기 위하여 암시적동기화가 수행된다.

- **조종동기화** 조종동기화조작을 실행하는 프로쎄스는 프로그람실행이 일정한 조종상태에 도달할 때까지 기다릴수 있다. 조종동기화의 보편적인 실례로서 차단동기화가 있는데 다음코드에서 보게 된다.

```
Parfor(i=1;   i<n;   i++){
      Pi
      Barrier
      Qi
}
```

여기에 n개의 프로쎄스가 있다. Pi를 수행하는 i째 프로쎄스다음에 barrier가 따르고 그다음 Qi가 놓인다. Pi를 끝내고 barrier명령문에 도달할 때 모든 다른 프로쎄스가 역시 그것을 barrier에 도달할 때까지 기다려야 한다. 또 다른 조종동기화구성은 림계범위인데 아래 실례에서 본다.

```
Parfor(i=1;   i<n;   i++){
      Critical{x=x+1;   y=y−1;}
}
```

림계범위는 서로 배타적이며 그안에서 오직 한 프로쎄스가 한 시각에 두 배정을 실행하게 한다. 대조적으로 다중프로쎄스들은 원자성이 집행되는 동안에 그것의 원자범위들을 실행할수 있다.

- **자료동기화** 자료동기화조작을 실행하는 프로쎄스는 프로그람실행이 일정한 자료상태들(2.2.1 참고)에 도달할 때까지 기다릴것이다. 실례로 wait(x>0)명령문을 실행하는 프로쎄스는 변수 x가 정의될 때까지 지연된다. 자료동기화조작의 실례들은 열쇠채우기, 조건부림계범위, 감시기사건들을 포함한다. 대부분 현재체계들에서 원자성은 자료동기화를 통하여 실현되는데 다음과 같이 실현된다.

```
Parfor(i=1;  i<n;  i++){lock(s)  ;   x=x+1;   y=y−1;  unlock(s);}
```

여기서 열쇠채우기동기화는 신호기발 s에 의하여 자료상태에 의존한다. 조종동기화는 일반적으로 자료동기화보다 더 쉬우며 수자가 더 융통성이 있다. 동기화는 7장과 14장에서 자세하게 론의할것이다.

집합 집합조작은 병렬프로그람의 요소프로쎄스들이 계산한 요소결과들을 혼합하여 완전한 결과를 발생시키는데 리용된다. 그것은 초기단계의 렬로써 실현되며 매개 첫 단계는 짧은 계산과 간단한 통신 또는 동기화로 구성되여 있다.

하나의 실례로 두 벡토르 A와 B의 스칼라적을 계산하는 다음프로그람을 고찰하자. 여기서 aggregate_sum은 부분합, $x[0]$, $x[1]$, ⋯ , $x[n-1]$은 통합하여 최종결과 inner_product=x[0]+x[1]+⋯+x[n−1]을 발생한다.

```
Parfor(i=0;   i<n;   i++){
```

```
    X[i]:=A[i]*B[i];
    Inner_product:=aggregate_sum(x[i];)
}
```

이 합연산을 **축소연산**이라고 부른다. 왜냐하면 여러 값들을 단일값으로 감소시키기 때문에 집합의 다른 형태로는 주사(역시 병렬앞붙이), 하강알고리듬, 상승알고리듬 [86_87, 518] 기타 등등이 있다.

실례 2.6 재귀적인 두배축소연산

*n*개의 프로세스 P(0), P(1),⋯, P(n−1)이 있다고 하자. 배렬요소 a[i]는 처음에 프로세스 P(i)에 분산된다. 축소 sum=a[1]+⋯+a[n−1]는 프로세스 P[i]를 위한 다음의 단일코드프로그람에 의하여 서술될수 있다. 여기서 i=0부터 n−1까지이다.

```
Sum=a[i]; //    매 프로세스는 국부변수의 합을 가진다.
For(j=1;   j<n;    j=j*2){ //    log(n)초걸음들
If(i%j=0{
프로세스 P(i*j)의 합을 국부변수 tmp에 넣기
sum=sum+tmp;
}
}
```

더하기는 *n*=8인 경우 그림 2−8에서 보는바와 같이 log *n* 첫 단계내에 나무형식으로 실현된다. 매 첫 단계에서 프로세스들은 다른 프로세스로부터 하나의 스칼라를 받고 중간부분합을 계산하기 위하여 스칼라더하기를 수행한다. 마지막에 프로세스 P(0)은 최종결과를 포함한다.

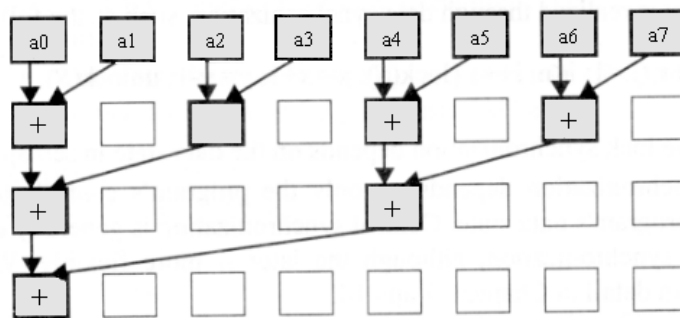



그림 2−8. 재귀적인 두배연산

이것은 집합의 주요한 특성이다. 즉 그것은 첫 단계의 렬로 이루어 지며 첫 단계안에서 매 프로세스는 계산의 적 grain을 수행하며 그다음 적은 통보를 통신한다.

2. 4. 2. 호상작용방식

그림 2-9를 고찰하자. 여기서 n개의 프로쎄스 $P_1, \cdots, P_n$은 호상작용코드 C를 실행하며 호상작용한다. N개 처리기들을 호상작용의 관계자(또는 참가자)들이라고 부른다. 코드 C가 모든 참가자들이 C에 도달했을 때까지 실행될수 없으면 **호상작용동기적**이라고 부른다. 처리기 C에 도달할 때 다른 프로쎄스들을 기다리지 않고 C를 수행하도록 전진할수 있을 때를 호상작용비동기적이라고 말한다.

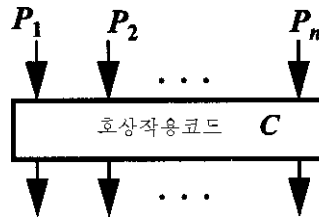



그림 2-9. n개 처리기사이 호상작용

n=2이면 호상작용을 두 관계자호상작용이라고 부르며 n이 2보다 크면 다중관계자호상작용이라고 부른다. 두 관계자통신을 **점대점통신**이라고 부르며 이때 한 프로쎄스는 또 다른 프로쎄스에로 통보를 보낸다. 다중호상작용은 특히 집합적호상작용이라고 부른다.

오직 동기호상작용을 가지는 병렬프로그람들은 리해하기 더 쉽다. 왜냐하면 호상작용코드 C가 실행될 때 모든 참가자프로쎄스들은 C에 도달하기때문이다. 다른 말로 C가 실행을 진행할 때 프로그람은 하나의 조종상태에 놓인다.

비동기호상작용에서 프로쎄스가 호상작용코드실행을 출발할 때 프로그람은 각이한 조종상태들에 놓일수 있다. 다른 프로쎄스들의 임의의것은 0에 도달하지 말아야 하는것, 도달해야 하는것, C를 통과한것이 있을수 있다. 그러나 비동기호상작용은 보다 다루기 쉬운 경향이 있다. 많은 병렬프로그람들은 오직 다중관계자동기호상작용만을 요구한다. 입력과 출력조건을 설정하는데 여러가지 방도가 있다. 프로쎄스 P가 호상작용코드 C에 직면할 때 많은 호상작용방식들이 가능하다. 프로쎄스 P의 다음상태는 호상작용코드 C에 관하여 정의될수 있다.

- **In** 프로쎄스 P가 코드 C에 있고 방금들어가기, C부분을 끝내기 또는 여전히 C를 수행하고 있는것을 포함한다.
- **Out** 프로쎄스 P가 C안에 없다.(도착하지 않았거나 탈출하였다.)
- **Arrived** 프로쎄스 P가 C에 방금 도착했지만 들어 가지는 않았다.
- **Finished** 프로쎄스 P가 C코드부분을 끝냈지만 아직 탈출하지 않았다.

두 관계자호상작용에 대하여 4^4 =256결합이 있을수 있다. 오직 실제적으로 몇개만이 사용된다. 일부 실례들을 표 2-3에서 보여 준다.

세가지 호상작용방식이 널리 사용된다.

표 2-3 입력과 출력조건의 결합

입력조건		출력조건		실례
자체	기타	자체	기타	
도착	X	In	X	보내기/받기차단
도착	X	끝	X	보내기차단
도착	X	끝	끝	받기차단
도착	도착	끝	끝	동기보내기/차단(CSP)
도착	도착	끝	In 혹은 끝	장벽
도착	Out	끝	Out	림계구역

- **동기적** 이 방식에서 모든 관계자들은 호상작용이 시작되기전에 도착해야 한다. 어떤 관계자프로쎄스는 호상작용을 탈출할수 있고 그다음 조작을 계속할수 있다. CSP에서 보내기/받기조작들은 하나의 실례이다. 호상작용이 동기적인가를 분간하는 간단한 방도는 두 장벽검사이다. 즉 한 차단은 호상작용코드앞에 놓고 다른것은 뒤에 놓는다. 결과 얻어 지는 병렬프로그람이 원래의것과 정확히 같은 의미론을 가진다면 호상작용은 동기적이다. 그렇지 않으면 비동기적이다.
- **차단** 다른 관계자들이 있는가 없는가에 관계없이 관계자는 그것이 도착하자마자 호상작용에 들어 갈수 있다. 그것은 호상작용이 부분적으로 끝났을 때 호상작용을 탈출할수 있다. 다른 관계자들은 호상작용코드에서 끝나지 않고서는 들어 갈수 없다는것을 알아야 한다. 차단은 하나의 실례이다. 그것의 완료는 문을 내보내지만 목적대상에서 반드시 받지 않는다는것을 의미한다.
- **비차단** 관계자는 도착하자마자 들어 갈수 있다. 그것은 호상작용의 부분을 끝내기전에 호상작용을 탈출할수 있다. 비차단은 하나의 실례이다. 그것의 완료는 프로쎄스가 보내기에 응답된다는것을 의미한다. 통보문은 반드시 내보내야 한다.

2. 4. 3. 호상작용패턴

호상작용패턴은 어느 관계자가 다른 어느 관계자에게 트랜잭션을 주는가를 의미한다. 그것의 형태가 콤파일시에 결정될 때 호상작용은 **정적**이라고 말하며 그렇지 않는 경우 동적이라고 말한다. 동적호상작용의 특수경우는 관계자그룹이 동적으로 변화될수 있을 때 일어 난다. 실례로 프로쎄스들은 실행시에 어떤 그룹을 탈퇴하거나 결합할수 있는데 이것은 동적그룹화로 알려 져 있다[261].

N관계자호상작용에서 형태가 프로쎄스첨수로서 간단한 기능으로 서술될수 있을 때 호상작용은 정규형태(regular)를 가진다고 말한다. 첨수 t와 j가 주어 졌을 때 i째 프로쎄스가 j째 프로쎄스에 영향을 주겠는지를 상수시간과 공간에서 결정될수 있는 알고리듬이 있다면 호상작용은 정규형태이다.

실례 2.7. 점대점과 집합통신모형

대역적인 정규통신이 그림 2-10에 제시되여 있는데 얼마나 많은 몇개의 송신자와 수신자가 통신에 포함되는가에 따라 다음과 같은 네개의 부류로 갈라 볼수 있다.

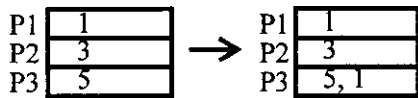
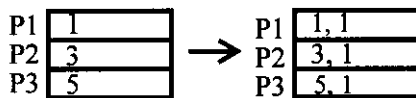
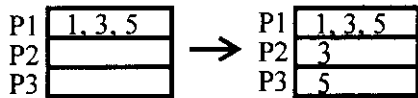
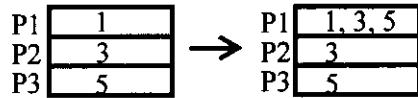
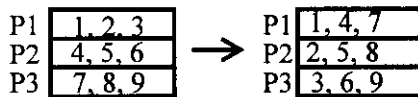
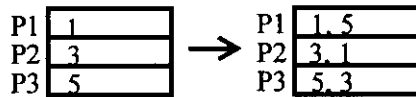
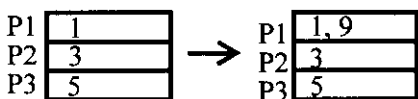
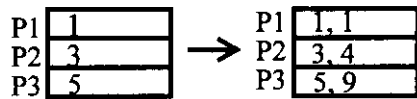
- **하나 대 하나** 이것은 점대점통신이라고도 한다. 그림 2-10 ㄱ)에서와 같이 하나의 송신자와 수신자가 있다.
- **하나 대 여러** 이것은 방송을(그림 2-10 ㄴ) 포함하는데 여기서 한 프로쎄스(뿌리라고 부르는)는 모든 프로쎄스들에 같은 통보문을 내보내는데 그자신도 포함된다. 산란조작(그림 2-10 ㄷ)은 뿌리가 각이한 프로쎄스에로 각이한(또는 개별화된) 통보문을 보낸다는 점에서 방송의 일반화이다.
- **여러 대 하나** 이것은 수집을 포함한다(그림 2-10 ㄹ). 여기서 뿌리프로쎄스는 구별된 통보를 매개 프로쎄스로부터 받는다. 전체로 뿌리는 *n*개 통보를 받으며 여기서 *n*은 그룹크기이다. 또 하나의 실례는 축소(그림 2-10 ㅅ)이다. 그것은 매개 프로쎄스로부터 하나씩 *n*개의 국부값을 뿌리프로쎄스의 하나의 최종값으로 모은다.
- **여러 대 여러** 가장 간단한 형식은 치환이다. 여기서 프로쎄스는 하나이상의 프로쎄스와 주고받기하지 않는다. 치환의 실례는 그림 2-10 ㅂ)에서 순환밀기이다. 또 하나의 실례는 주사(그림 2-10 ㅇ)이다. 그것은 *n*개의 국부값을 *n*개의 최종값으로 종합하는 계산에 의하여 축소를 일반화한다. 전체적교환(그림 2-10 ㅁ)에서 매개 프로쎄스는 개별화된 통보를 *n*개 처리기의 매개에로 보내는데 그자체도 포함된다.

ㄱ) 점대점: P1은 1을 P3으로 보낸다

ㄴ) 방송: P1은 1을 모두에게 보낸다

ㄷ) 산란: P1은 한개값을 매 마디에 보낸다

ㄹ) 수집: P1은 매 마디로부터 한개 값을 얻는다

ㅁ) 전체교환: 매 마디는 구별되는 통보문을 매 마디에 보낸다

ㅂ) 밀기: 매 마디는 한개 값을 다음 마디에 보내며 이전마디로부터 값을 받는다

ㅅ) 감소: P1은 합 **1 + 3 + 5 = 9**을 얻는다

ㅇ) 주사: P1은 1을 얻고 P2은 **1 + 3 = 4**, 를 얻으며 P3은 **1 + 3 + 5 = 9**를 얻는다.

그림 2-10. 점대점통신연산과 집중통신연산

사용자가 호상작용방식에서 어떤 구속을 받지 않는 동안 (바꾸어 말하면 동기 또는 비동기성) 호상작용의 형태들은 보통 응용에 의존된다. 병렬언어는 보통 동적 및 비정규 호상작용패턴들에 대하여 제공된다. 비정규통신의 한 부류는 정확한 통신형태를 모를 때 *h*−관계(정의 1.6참고)이다.

매개 마디는 최대로 *h*단어들을 여러 마디들에 보내며 매 마디는 최대로 *h*단어들을 받는다. 통신형태를 모르면 통신알고리듬은 최악의 경우에 대처해야 한다. 왜냐하면 대부분의 *h*−관계알고리듬은 전체적교환에 기초하기때문이다.

2．4．4．협동호상작용과 경쟁호상작용

이 책에서 병렬프로그람에 대하여 말할 때 초보적으로 병렬응용프로그람 즉 사용자(프로그람작성자)가 어떤 병렬콤퓨터우에서 실행되게 작성한 응용프로그람을 개발하는것과 관계된다. 이러한 프로그람들은 체계프로그람(실례로 병행조작체계)과 다르다.

하나의 주요한 차이는 프로세스들이 여러가지 방식에 따라 호상작용한다는것이다. 병렬조작체계에서 프로세스들은 주로 공유자원들에 대하여 경쟁하면서 호상작용한다. 우리는 이러한 호상작용을 **경쟁적인 호상작용**이라고 부른다. 다른 한편 어떤 함수를 계산하는데서 병렬응용프로그람들에 있는 프로세스들이 협조하는 경우가 있다는것이다. 이 프로세스들이 호상작용할 때 기본문제는 하나의 자원을 안전하게 공유하지 못하는것이다.

오히려 그것들은 정보를 서로 통신하여 서로 동기화하거나 부분적으로 계산된 값들로부터 결합한 값을 발생하려고 한다. 경쟁호상작용들은 보통 프로그람의 자료상태에 의존하고 동기화조작을 쓸것을 요구하며 한편 협조동기화조작은 오직 프로그람의 조종상태에만 관계한다.

표 2-4　　　　　경쟁 대 협동 병렬프로그람

속성	경쟁프로그람	협동프로그람
응용실례	조작체계망도구	편미분방정식, 리산사건모의
립도	A Few Large−Grain Heterogeneous Processes	Many Small−Grain Homogeneous Processes
호상작용실례	림계구역, 생산자−소비자, 독자−필자,	장벽동기화, 방송, 치환행 렬축소, 주사, 경사 work pool 혹은 대기
호상작용에 대한 구조	차단, 신호기, 사건, 조건림 계구역, 모니터	장벽, 꺼내기−첨가 일치성구역
성질	비종단, 미정	종단, 확정

사용자는 일반적으로 그것의 병렬계산프로그람을 끝낼수 있고 결정적인것을 기대한다(즉 프로그람들은 중지하고 결과는 유일해야 한다). 이와는 반대로 병렬조작체계코드는 보통 비종결, 비결정적이다. 프로그람의 두 형태에 대한 몇가지 중요한 차이들이 표 2−4

에서 간단히 제시한다.

력사적으로 경쟁호상작용들은 훨씬 더 많은 관심을 끌었으며 많은 구성이 제안되였는데 그것은 차단, 신호기, 사건, 조건부림계구역, 감시기[37]와 같은것이다. 다만 몇년전에 많은 다중처리기에서 사용된 프로그람작성언어들은 오직 경쟁호상작용들에 대한 구성만을 제공하였다.

이러한 언어에서 우선 협조적호상작용은 자원공유호상작용에로 변환될(사용자에 의하여) 필요가 있으며 그다음 경쟁호상작용프리미티브들을 통하여 실현될것을 요구한다. 이러한 실현에 대한 방도는 효과적이 못될뿐아니라 또한 리해하고 검사수정하기도 어렵다. 이 책의 주되는 목적은 장벽과 4편에서 론의되는 일관성령역과 같은 구성을 연구하는것이다. 이 구성들은 특히 협조호상작용을 위해 설계된다.

2. 5. 병렬프로그람의 의미론

의미문제는 병렬프로그람의 다양한 거동적특성을 가리킨다. 여기서는 두가지 문제 즉 종결성과 결정성만을 론의한다. 이 문제들은 병렬프로그람으로 하여금 언제나 끝내고 결정하게 하므로 중요하다. 사실상 계산을 진행하는 대부분응용들가운데서 동일한 점은 물리적세계를 모형화하는것이다. 계산물리학, 계산화학, 계산생물학 등은 그것의 좋은 실례로 된다.

병렬계산이 과학실험에서 효과 있게 되자면 그것은 두개의 기본계산속성 즉 종결성과 결정성을 가져야 한다. 그 목적은 유한시간에 끝나는것을 담보하고 계산실험이 다른 사용자들에 의하여 반복될수 있게 하는것이다.

2. 5. 1. 프로그람종결

종결문제는 프로그람을 표준적으로 종결하든가 그렇지 않으면 비정지의 가능한 리유를 취급한다. 비종결에는 세가지 형태들이 있는데 그것은 무한순환, livelock와 교착이다.

정의 2. 5 프로그람이 항상 실행을 끝내고 임의의 초기상태로부터 최종상태에로 들어간다면 종결이다. 어떤 초기상태로부터 정지함이 없이 련속적인 실행을 하는 상태에 들어 갈수 있으면 프로그람은 발산한다. 프로그람이 최종상태에 있지 않고 련속실행함이 없이 실행도중에 멈추어 서 교착한다.

비종결성은 순차프로그람과 병렬프로그람에 서로 각이하게 나타난다. 순차프로그람은 교착되지 않는다. 따라서 순차프로그람의 실례는 끝나지 않으면 그것은 발산한다. 발산하는 실례적인 순차프로그람은 무한순환 " while(x>0 x:=x+1) "이다. 병렬프로그람에서 발산의 두번째 형태가 있는데 그것을 Livelock라고 부르며 거기서 몇가지 프로쎄스들이 호상간섭으로 하여 무한순환에 빠진다. Livelock의 간단한 실례는 다음의 코드이다.

Par{

```
while(y>0)  x:=x+1;
while(x>0)  y:=y+1;
}
```

병렬프로그람은 교착으로 하여 비종결일수 있다. 콤파일러나 실행시 체계는 모든 비정지상황을 탐지해야 한다. 이 문제는 튜링기계문제와 등가이며 비결정성이다. 따라서 병렬언어가 교착과 Livelock를 방지하거나 콤파일러와 실행시 체계가 그것들을 탐지하게 하도록 설계하는것이 중요하다.

이런 견지에서 암시적병렬성을 리용하는 프로그람작성체계는 고급하다. 왜냐하면 이 언어는 데드로크와 리브로크가 발생하지 않는다는것을 담보하기때문이다.

2. 5. 2. 프로그람결정성

우선 두 개념 즉 결정성과 결정론의 차이를 구별할 필요가 있다. 결정론은 유일한 계산(바꾸어 말하면 원자조작들의 유일한 렬)을 의미하고 결정성은 유일한 최종결과를 의미한다. 또한 결정론의 정의는 프로그람이 같은 초기상태를 가지고 여러번 수행된다면 이 매개의 각이한 실행에서 같은 유일한 계산렬이 얻어 진다는것을 의미한다.

결정성의 정의는 같은 초기상태로부터 각이한 실행에 대하여 최종결과는 같다는것을 의미한다. 약속에 기초하는 순차프로그람들은 대부분 결정성을 가지며 따라서 확정적이다. 그러나 순차프로그람들에서 비결정성은 미지의 현상은 아니다. 어떤 변수에 우연수발생수를 통하여 값이 배정될 때 값의 초기자료상태를 가지는 순차프로그람은 각이한 실행에서 각이한 결과들을 얻을수 있다.

그러나 비결정성의 이러한 형태는 자주 Monte Carlo모의에서 또는 유전알고리듬에서 사용자가 기대하는것이다. 다른 말로 이 응용들은 본질적으로 확정적이 아니다. 많은 응용의 계산함수들은 수학적으로 확정적이다. 그러나 이 함수들을 실현하는 병렬프로그람들은 병렬실행의 결과로써 미지로 된다. 미지의 이런 례외적인 형태를 찾아 내고 취급해야 한다. 또한 암시적병렬성을 리용하는 프로그람작성체계들은 높은 수준이다. 왜냐하면 이 언어는 모든 프로그람이 확정적이라는것을 담보하기때문이다.

2. 6. 참고문헌주해와 련습문제

병렬프로그람작성의 중요성과 유감스러운 문제에 대하여 지금까지 많이 지적되였다. 참고문헌[443,556,587]은 그들의 몇가지 견해들과 이 분야에서의 합의들을 개괄한다. Pancake와 조수들은 왜 병렬프로그람작성이 어려운가 하는것을 분석하였다[481,482]. 난점들을 고찰할 때 Dfiater는 순차프로그람병렬프로쎄스(SPPP)양상(실례로 시한무프로쎄스량프로쎄스)[497]을 주장하였다.

모두가 현재 병렬프로그람작성환경들의 대부분이 개선될 필요가 있다고 한것과 같이 무엇이 가장 좋은 방도이며 또는 좋은 방도가 있는가 하는 합의는 없다. 그렇지만 대부분의 사용자, 판매자, 연구자들로 구성된 병렬위원회는 네가지 긴요한 프로그람작성모형에로 의견일

치를 보았는데 그에 대하여 4편에서 론의한다.

다른 연구자들은 여전히 강하게 자료흐름과 함수적인 프로그람작성모형[123]과 같은 긴요하지 않은 양상들을 강하게 주장하고 있다[123]. 병렬대상지향프로그람작성에서 최근의 진보는 문헌 [58,139,144]에서 론의된다. 비필수적프로그람작성은 이 책의 범위를 벗어 난다.

프로쎄스와 스레드의 개념은 조작체계의 견지에서 지금까지 문헌 [582,621]에서 론의되였다.

IEEE Dthreads표준이[345] 출판되였다. 대부분의 현존조작체계들은 스레드를 지원하는데 Alx에 대하여 문헌 [340]에서, Solards에 대하여 문헌 [132,601]에서, Digital Unix에 대하여 문헌 [193]에서 서술하였다.

병렬성들을 서술하기 위하여 다양한 언어표기법들이 Andrews에 의하여 개괄되였다[37]. 단일코드와 다중코드도식들은 특허권이 없어 진 체계(실례로 PVM[261])와 상업적체계들(실례로 IBM SP2[337])에서 사용되여 왔다. 자료병렬방법은 더우기 14장에서 론의될것이다. 구역분할과 일정작성기문제는 문헌 [230]에서 개괄하였다.

병렬프로그람들은 순차프로그람보다 훨씬 더 복잡한 의미를 가진다. Pobert Keller의 독창적인 연구론문[366]은 우선 표시화된 이행체계개념을 제안하였는데 그것은 대부분 조작의미모형의 토대이다. Apt와 Olderug의 책 [41]은 구성방식화된 조작의미들에 대한 좋은 소개로 된다. 대수적의미론은 Baeten과 Weijland[53], Milner[449]에 의해서 론의되였다.

문 제

문제 2.1. 프로쎄스 *P*가 두개 스레드 *t*와 *u*를 가진다고 가정하자. *T*와 *u*는 *P*의 주소공간을 공유하지만 그 매개는 비공개탄창을 가지고 있다. *T*와 *u*가 비공개복사를 가지는 세가지 다른 자료와 조종구성방식들을 렬거하시오.

문제 2.2. 그림 2−3 ㄱ)에서와 같이 조종변수와 자료변수를 렬거하시오. 그림 2−3 ㄴ)와 그림 2−3 ㄷ)에 대하여 같은것을 취하시오. 프로그람변수들에서 그것들의 차이에 대하여 설명하시오.

문제 2.3. 그림 2−3을 써서 표 2−1을 설명하시오.

- 그림 2−3 ㄷ)에 대한 그림 2−3 ㄴ)의 우점은 무엇인가?
- 그림 2−3 ㄴ)에 대한 그림 2−3 ㄷ)의 우점은 무엇인가?
- 그림 2−3 ㄹ)는 그림 2−3 ㄴ), ㄷ)의 절충이라고 하는가?

문제 2.4. 프로쎄스개념과 관련한 다음의 용어들을 구별하시오.

- 방식절환, 문맥절환, 프로쎄스절환
- 프로쎄스상태, 프로그람상태

- 자료상태와 조종상태
- 프로쎄스, 과제, 스레드

문제 2.5. 최근에 일부 연구자들은 단일주소공간조작체계를 주장하였다. 그 사상은 모든 프로쎄스와 핵심부가 같은 주소공간에 놓인다는것이다. HTTP데몬, DNS데몬과 Web봉사콤퓨터우에서 실행된다고 하자.

- Unix우에서 단일주소공간방법의 주요우점을 설명하시오.
- 단일주소공간에 대한 주요 능력검사문제들을 설명하고 그 풀이를 제안하시오.

문제 2.6. 프로쎄스일정작성기와 관련한 다음의 용어들을 구별하시오.

- 단순단일프로그람화프로쎄스
- 단일프로그람화된 묶음프로쎄스
- 다중프로그람화된 묶음프로쎄스
- 시분할프로쎄스
- 독점다중프로그람화된 프로쎄스

문제 2.7. 실례 2.2를 고찰하고 다음질문에 답변하시오. 경과시간은 실행의 매 프로쎄스를 허락한 때로부터 실행이 완결될 때까지의 시간이다.

- 호상작용프로쎄스에 대하여 어떻게 900s의 경과시간이 나오는가?
- 어떻게 시분할경우 50s의 경과시간이 나오는가?
- 호상작용체계가 훨씬 높은 우선권을 가지고 독점일정짜기기능이 있는 체계에서 수행된다면 경과시간은 무엇인가?

문제 2. 8. 병렬성문제에 대한 다음의 용어들을 구별하고 실례를 쓰시오.

- SPMD와 MPMD
- 암시적병렬성과 명시적병렬성
- 정적병렬성과 동적병렬성
- 자료병렬성과 조종병렬성

문제 2.9. 그림 2-3의 세 병렬코드들을 고찰하시오. 그 매개에 대하여 다음 질문에 답변하시오.

- 코드가 SMD인가 또는 MDMP인가 ?
- 코드가 암시적병렬성을 쓰는가 명시적병렬성을 쓰는가 ?
- 코드가 정적병렬성을 쓰는가 동적병렬성을 쓰는가 ?

- 코드가 자료병렬성을 쓰는가, 조종병렬성을 쓰는가 ?

문제 2. 10. 호상작용문제에 관한 다음용어들을 구별하고 실례를 드시오.

- 자료동기화와 조종동기화
- 경쟁호상작용과 협조형호상작용
- 정적호상작용과 동적호상작용
- 점대점통신과 집합통신
- 정규통신과 비정규통신

문제 2.11. 그림 2−10 ㄱ)로부터 그림 2−10 ㅂ)까지 고찰하시오. 그 매개에 대하여 *n* 관계가 통신인가를 설명하시오(실례로 왜 1관계인가, 5관계인가?)

문제 2.12. 주사는 감소의 일반화이다(실례 2.6을 참고). N프로쎄스가 있고 프로쎄스 P ㅈ)가 a[i]로 표시되는 값을 포함한다고 가정하자. 그러면 한번 주사후에 프로쎄스 P ㅈ)는 결과 a[i]+⋯+a[n−1]을 얻을것이다.

- 짧은 단일코드프로그람이 주사를 계산하는지 보시오. 주사조작은 log *n* 첫 단계에서 실현된다.
- 어떻게 프로그람이 *n*=8에 대한 주사를 계산하는가를 례증하는 그림을 보시오. 그림 2−8의 양상을 따르시오.

문제 2. 13. 다음의 코드가 옹근수배렬 A의 합을 계산하는가를 고찰하시오.

```
Sum=0
For(i=0;   i<N;   i++)Sum=sum+A[i];
```

- parfor와 Atomic범위구성을 사용하여 기지적인 합함수를 계산하는 병렬프로그람을 쓰시오. 배렬의 크기는 N이고 처리기의 수는 *n*이다. N은 *n*을 나눈다.
- Parfor와 aggregation구성을 써서 part를 반복하시오.
- 간단성과 성능에 의하여 이 두 프로그람을 비교하시오.
- 이미 알려 진 코드가 실제적인 병렬콤퓨터우에서 실행될 때 미지의 상황이 있는가

문제 2.14. 다음의 방법들을 사용하여 옹근수배렬의 합을 계산하는(문제 2.13) 간단한 SMMD병렬프로그람을 쓰시오. 명확히 정의된 임의의 허위코드표기법을 쓸수 있다.

- 서고방법
- 새로운 구성방법
- 콤파일러지시어방법

제3장. 성능척도와 성능평가기준

확대가능성체계들은 콤퓨터들에서 높은 성능에 대한 끊임없이 증가되는 요구를 만족하도록 만들어 져야 한다. 이 목표에 도달하기 위해서 콤퓨터사용자들과 설계자들은 다음과 같은 성능상 문제들을 고찰해야 한다.

- 응용프로그람과 체계의 성능을 어떻게 특징 짓는가 하는 일반적인 방법은 성능평가법으로서 3.1에서 론의된다.
- 가격대성능비에서 사용자의 요구는 무엇인가 하는 문제는 3.2에서 취급한다.
- 이 두 요구의 절충은 실행시간, 지속속도, 비용효과성 등에 의하여 론의된다.
- 어떻게 응용프로그람의 성능을 측정해야 하며 어떤 종류의 성능척도가 사용되여야 하는가 하는 문제에 대하여 3.3에서는 순차프로그람, 3.5에서는 병렬프로그람을 서술한다.
- 병렬프로그람이 병렬콤퓨터에서 실행될 때 체계성능은 어떻게 특징 지어야 하며 성능에 영향을 주는 인자들은 무엇이고 또 현재 체계들에서 전형적인 인자들은 무엇인가 하는 문제들은 3.4에서 고찰한다.
- 체계확대가능성은 어떻게 량적으로 규정하고 분석하며 주어 진 응용프로그람을 실행하는 병렬콤퓨터의 확대가능성은 어떻게 결정할수 있는가 하는 문제는 3.6에서 기본성능법칙으로 고찰한다.

3. 1. 체계와 응용의 성능평가기준

성능평가기준은 어떤 클라스의 응용프로그람에 대한 처리와 자료이동특성을 가상적으로 얻어 내는 성능검사프로그람이다. 성능평가기준은 콤퓨터의 성능을 측정하고 예측하여 그것의 구조적약점과 우점들을 발견하는데 사용된다. 성능평가기준묶음은 검사조건과 절차를 관리하는 특정한 규칙모임을 가진 성능평가기준프로그람의 모임으로서 검사되는 가동환경, 입력자료, 출구결과들, 성능척도들을 포함한다. 성능평가기준계렬은 성능평가기준묶음들로 이루어 진 하나의 모임이다.

성능평가기준분류 성능평가기준은 응용프로그람클라스에 따라 과학계산, 상업적응용, 망봉사, 다매체응용프로그람, 신호처리 등과 같이 분류한다. 성능평가기준들은 또한 마크로성능평가기준과 마이크로성능평가기준으로 나눌수 있다.

마크로성능평가기준들은 콤퓨터체계의 성능을 하나의 전체로서 측정한다. 그것은 응용프로그람클라스에 대해 각이한 체계들을 비교하므로 체계구매자에게 리용가치가 있다. 그러나 마이크로성능평가기준은 매 체계가 잘 또는 나쁘게 수행하는가를 발견하지 못한다. 마이크로성능평가기준은 CPU속도, 기억속도, I/O속도, 조작체계성능, 망기능 등의 성질을 평가한다.

성능평가기준은 충분히 독립적인 응용프로그람이거나 핵심부일수 있다. 그것은 기본

특성을 유지하고 있는 응용프로그람으로부터 얻어 진 훨씬 간단하고 단순한 프로그람이다. 성능평가기준은 성능평가기준기능을 위하여 특수하게 설계된 합성프로그람 또는 실제적인 작업을 하는 프로그람일수 있다. 마이크로성능평가기준은 종합적인 특성을 가진다. 지금까지 100개를 넘는 성능평가기준묶음들이 제기되여 리용하고 있다. 몇가지 일반적인 성능평가기준묶음들이 표 3-1에 렬거된다.

표 3-1 대표적인 마이크로와 마크로성능평가기준

형태	이름	측정
마이크로 성능평가기준	LINPACK	수값계산(선형대수)
	LMBENCH	Unix에서 체계호출과 자료이동연산
	STREAM	기억기대역너비
마크로 성능평가기준	NAS	병렬계산(CFD)
	PARKBENCH	병렬계산
	SPEC	혼합용성능평가계렬
	Splash	병렬계산
	STAP	신호처리
	TPC	상업적응용

3. 1. 1. 마이크로성능평가기준

아래에서 세개의 마이크로성능평가기준묶음을 간단히 소개한다. 자세한것은 관련조직들에서 제출한 개별적인 조종시험문서에서 알수 있다.

LINPACK성능평가기준은 Tennessee종합대학에 있는 Jock Dogarra에 의하여 만들어졌으며 리용되고 있다. 그것은 하나의 성능평가기준이라고 부르지만 LINPACK는 실제적인 계산작업을 위해서 널리 사용된다. 그것은 선형방정식과 선형최소평방문제들을 해석하고 분석하는 Fortran부분프로그람집합체이다. 그 행렬은 속박되고 대칭무한,

표 3-2 1996년 12월 LINPACK기록표본

콤퓨터	처리기개수	R_{max} (Gflop/s)	N_{max} (order)	$N_{1/2}$ (order)	R_{peak} (Gflop/s)
Intel ASCI Option Red	7264	1068	215,000	53,400	1453
CP-PACS	2048	368.2	103,680	30,720	614
Intel Paragon XP/S MP	6768	281.1	128,600	25,700	338
Numerical Wind Tunnel	167	229.7	66,132	18,018	281
Fujitsu VPP500/153	153	200.6	62,730	17,000	245
Cray T3D 1024	1024	100.5	81,920	10,224	152
IBM SP2-T2	512	88.4	73,500	20,150	136
NEC SX-4/32	32	66.53	15,360	1792	64

대칭정의값일수 있으며 삼각행렬, 세줄대각선정방행렬일수 있다. 그것은 체계의 수값계산가능성의 하나의 좋은 확대로서 사용하기 쉽고 간단하다. LINPACK에 가까운 방법들은 여러가지 체계의 LINPACK성능수들이 정기적으로 출판하는데 500개의 목록을 포함한다. 그것은 세계적으로 500개의 매우 강력한 콤퓨터들이다. LINPACK는 분산기억병렬콤퓨터들에서 ScaLINPACK로 변경되였다. 꽉대기 8개의 대단히 강력한 콤퓨터의 LINPACK성능표본을 표 3-2에서 보여 주었다. 여기서 R_{max}는 달성된 지속적인 최대속도이고 N_{max}는 R_{max}에 도달했을 때 문제크기이다. 바꾸어 말하여 자료행렬의 차수 $N_{1/2}$은 R_{max}가 절반에 도달했을 때 문제의 크기이다. R_{peak}는 측정된 체계의 최대속도이다.

1996년 12월에 제작된 지금까지 제일 빠른 콤퓨터는 7264처리기 Intel OptionRed이며 그것은 1453Gflop/s의 최대속도를 가진다. 1.068Tflop/s의 지속속도는 215,000×215,000행렬로 특징지어 지는 문제를 풀어서 얻어 진다.

LMBENCH LMBENCH성능평가기준묶음은 SGI의 Larry MoVoy에 의하여 보존되고 있다. 그것은 조작체계부가처리와 여러가지 UNIX가동환경우에 있는 처리기, 캐쉬, 기억, 망과 디스크사이에서 자료전송능력을 측정하는데 쓰이는 이식가능한 성능평가기준이다. 그것은 성능문제와 체계설계를 식별하기 위한 간단하고 아주 쓸모 있는 도구이다. 3개 체계에 대한 몇가지 LMBENCH결과들을 표 3-3에서 보여 준다. 3개체계는 IBM990봉사기, Sun Ultra워크스테이션, Intel Adler PC체계이다.

표 3-3 LMBENCH에 의해 측정된 대역너비, 지연시간, 체계부가처리

속 성		Intel Alder	Sun Ultra	IBM 990
대역너비 (MB/s)	기억기복사	52	85	242
	파일읽기	52	85	187
	판	38	61	84
	TCP	20	51	
지연시간 (μs)	기억기복사	0.28	0.27	0.26
	파일읽기	23,809	18,181	13,333
	판	101	62	91
	TCP	305	162	332
체계부가처리 (μs)	빈값체계호출	7	5	16
	프로쎄스창조	4500	3700	1200
	문맥절환	36	14	13

STREAM STREAM성능평가기준은 SGI의 John Mocalpin이 보존하는 간단한 종합성능평가기준으로서 지속적인 기억기대역너비(MB/s)와 대응하는 계산률들을 측정한다. STREAM성능평가기준을 개발하기 위한 동기는 처리기들이 기억기보다 훨씬 더 빠르고 많은 프로그람들이 처리기속도보다는 기억기대역너비의 성능에 의하여 제한된것과 관련되여 있다.

이 성능평가기준은 단위걸음자료접근을 가진 몇가지 호상작용에 대하여 표 3-4에서 보여 준 네가지 조작들을 수행한다(바꾸어 말하면 매 호상작용마다 첨수는 1씩 증가한다.).

기억읽기와 쓰기는 대역너비를 계산하는데 포함된다. 성능평가기준은 적합한 캐쉬보다 훨씬 더 큰 자료모임을 가지고 동작할수 있게 설계한다.

표 3-4의 *a*, *b*, *c*벡토르들은 2백만개 요소배렬이며 하나의 요소는 8B단어이다. McCalpin은 기계균형척도를 제기하였는데 다음과 같다.

$$기계균형 = \frac{최대류동소수점속도(flop/s)}{지속\ TRIAD기억대역너비(단어/s)} \quad (3.1)$$

표 3-4 STREAM성능평가기준에서 4개의 연산

이름	코드	Bytes/반복	Flop/반복
COPY	a(i)=b(i)	16	0
SCALE	a(i)=q×b(i)	16	1
SUM	a(i)=b(i)+c(i)	24	1
TRIAD	a(i)=b(i)+q×c(i)	24	2

표 3-5 기계균형값에서 력사적추세

년도	체계	기억기대역너비 (MB/s)	최대속도 (Mflop/s)	기계균형
1978	DEC VAX 11/780	4	0.4	0.8
1991	DEC 5000/200	28	10	2.9
1993	DEC 3000/500	100	150	12
1995	DEC 600-5/300	169	600	28.4
1995	DC 8400/350	234	700	24
1980	IBM PC 8088/87	2	0.1	0.2
1992	IBM PC 486/EC2-66	33	10	2.4
1994	IBM PC Pentium-100	85	66.7	6.3
1990	IBM RS/6000-320	60	40	5.3
1993	IBM RS/6000-580	240	126	4.2
1994	IBM RS/6000-590	654	262	3.2
1995	IBM RS/6000-591	800	310	3.1
1989	SGI 4D/25	13	8	5
1992	SGI Crimson	62	50	6.5
1993	SGI Challenge	57	75	10.5
1994	SGI Power Challenge	135	300	17.8
1996	SGI Origin 200	317	388	9.8

기계균형척도는 한 단어를 읽고 쓰기하는 시간주기내에 실행될수 있는 flop수로서 해석될수 있다.

표 3-5에서 례증되는것처럼 많은 체계들의 기계균형값은 해마다 증가하고 있으며 기억기대역너비는 처리기속도보다 점점 더 많이 뒤떨어 진다는것을 의미한다.

례외적인것으로서 IBM RS/6000봉사기들인데 그것은 기억기체계설계에 커다란 주의

를 돌리였다. 최근년간에 다른 회사들도 역시 기억기체계성능을 개선하려고 노력하였다(실례로 DEC8400과 SGI Origin 200).

3. 1. 2. 병렬계산성능평가기준

많은 병렬계산성능평가기준묶음들이 쓰이고 있다. 실례로 Splash와 Splash-2는 Stanford University에서 개발된 수값계산성능평가기준들인데 분산공유기억기기계들을 [649] 연구하는데 널리 쓰이였다. Minols종합대학에서 개발된 완전한 성능평가기준묶음은 여러가지 병렬성콤파일러체계들과 기술들을 평가하는데 사용되여 왔다[76, 91, 176].

아래에서 세가지 병렬실험조들인 NPB, PARBENCH와 과학계산에서 주요응용그룹들을 표현하는 STAP에 대하여 서술한다.

NPB조 NAS병렬성능평가기준(NPB)은 NASA Ames Research Center에서 수값공기력학모의(NAS)프로그람에 의하여 병렬고속콤퓨터들의 성능평가를 위하여 개발되였다. NPB는 대규모계산류체동력학(CFD)응용프로그람들의 계산과 자료이동특성을 모방하였다.

NPB는 MPI-토대 Fortran원천코드실현뿐아니라 연필과 종이특성을 제공한다. 성능평가기준결과들은 NAS에 의하여 검증되고 정기 NAS보고서에 출판되였다. 이 특징으로 하여 NPB는 병렬콤퓨터판매자들, 사용자들, 연구자들속에서 좋은 호평을 받았다.

NPB조는 다섯개의 핵심부[EP,MG, CG, FT, IS]로 구성되여 있으며 세개의 모의응용(LU,SP,BT)프로그람들로 구성된다. 계산의 대부분은 IS에서 옹근수산수이며 한편 다른 성능평가기준들은 류동소수점계산집약적이다. EP(Embarrassingly 병렬)성능평가기준은 적은 통신을 가지는 몇개의 처리기들에서 실행될수 있으므로 적절하게 이름을 붙였다. 그것은 병렬콤퓨터의 류동소수점성능을 위한 도달가능한 웃한계를 평가한다. IS(옹근수 정돈)성능평가기준은 파케트정돈에 기초한 병렬정돈프로그람이다. 그것은 많은 량의 전체적인 교환통신을 요구한다.

MG(다중격자법)성능평가기준은 3차원스칼라빠쏭방정식을 푼다. 그것은 짧고 긴 령역통신을 둘다 수행하는데 그 통신들은 고도로 구조화된다. CG(공액트라디엔트법)성능평가기준은 대칭정값행렬의 최소고유값을 계산한다. 그것은 구조화되지 않은 격자계산특징이 있고 비정규적인 긴 령역통신을 요구한다.

FT성능평가기준은 3원편미분방정식을 FFT에 기초한 스펙트르법을 사용하여 풀며 역시 긴 통신을 요구한다. BT(블로크, 3 대각), LU(블로크아래 3각, 블로크웃 3각) 그리고 SP(스칼라 5 대각)성능평가기준들은 Navier Stokes방정식을 푸는데 여러가지 방법을 적용한다. LU는 많은 수의 작은 통신(매개는 다섯개 단어들)을 수행한다. 한편 다른 두개는 coarse-grain통신들을 사용한다.

PARKBENCH PARKBENCH(PARallel Kernels and BENCHmarks)위원회를 고속계산 92에서 병렬콤퓨터성능평가기준기술에 관심 있는 사람들에 의하여 설립되였다. 그 그룹이 제기한것은 성능척도와 표기법의 무모순적모임을 확립한것으로써 이 책에서 서술하였다.

현재 성능평가기준은 분산기억다중콤퓨터들을 위한것이다. 그것은 Fortran 77더하기와 통보교환을 위한 PVM 또는 MPI를 가지고 코드화된다. 문제는 공유기억기구조에 대한

Fortran 90과 HDF변종, 성능평가기준을 개발하는데 있다. 이 그룹은 지금까지 다음과 같은 네가지 종류의 성능평가기준을 내놓았다.

- **낮은 준위성능평가기준** 이 주소성능평가기준은 시계측정기분해능, 산수연산속도, 캐쉬와 기억기속도, 통신시동시간과 대역너비, 동기화부가처리와 같은 기본파라메터들을 측정한다.
- **핵심부성능평가기준** 이러한 측정부분프로그람들은 빈번히 행렬연산, FFT와 같은 과학계산과 편미방과 NPB핵심부들을 푸는데 리용된다.
- **콤팍트응용프로그람성능평가기준** 현재 병렬스펙트르변환, 얕은물모형응용프로그람과 세개의 NPB모의응용프로그람들만을 포함한다.
- **HPF콤파일러성능평가기준** 이것들은 HPF콤파일러의 성능을 측정하는데 리용된 몇가지 간단한 종합응용프로그람들이며 명시적인 HPF구성들에 대한 병렬실현에 초점을 두고 있다.

병렬 STAD조 시공간적용처리(STAP)성능평가기준묶음은 실시간탐지기신호처리성능평가기준프로그람의 모임으로서 원래 Mit Lincoln Laboratory[101]에서 개발되였다. MIT의 순차형STAD는 최근에 University Southern California에서 여러가지 MPP들을 평가하기 위하여 병렬STAP로 변환되였다. Hwang들이 그에 대하여 제기하였다[334,634].

STAP성능평가기준프로그람들은 계산이 집약적이며 그것은 초단위로 자료의 $O(10^2–10^4)$MB에 대해 $O(10^2–10^4)$류동소수점연산수행을 요구한다.

STAP성능평가기준묶음은 다섯개의 프로그람 즉 적응처리시험대(APT), 고차 POST_Doppler(HO_PD), Beam Space PRI_Staggered Post Doppler(BM_Stag), 요소공간 PRI_Stagered Post Doppler(EL_Stag)와 General(GEN)로 구성된다. GEN프로그람은 정돈(SORT), 고속푸리예변환(FFT), 벡토르적(VEC), 선형대수(LA)를 실행하는 네개의 독립요소프로그람들로 이루어 져 있고 그것은 탐지기신호처리응용들에서 리용되는 핵심부 부분프로그람들을 표현한다.

다른 네개의 성능평가기준들은 모두 도플러처리(DP)단계로부터 출발하는데 이 단계에서 프로그람은 기대하는 많은 량의 1차원FFT계산을 수행한다. 모든 네개의 프로그람들은 목표검출(TD)단계에서 끝난다.

APT는 삼각학습형행렬을 발생하기 위해 HouseHolder변환을 수행하며 그것은 그후의 전파장애와 혼란을 제거하는 복사형성(BF)단계에서 리용된다. 이와는 반대로 HO_PD프로그람에서 두개의 적응복사형성단계는 한 단계로 결합된다.

BM_Stage프로그람과 EL_Stag프로그람은 HO_PD와 류사하지만 각각 전파복사공간과 요소공간에서 알고리듬을 세련시키는 서로 엇갈린 호상간섭을 리용한다.

실례 3.1 ADP성능평가기준프로그람

STAP성능평가기준묶음에서 ADP프로그람은 다음과 같이 간단하게 서술할수 있다. 여기서 변수 N는 문제의 파라메터이다. 표기 [·]는 그 차원에 따라 모든 배렬요소들이 접근될수 있다는것을 의미한다. 변수 house는 N과 무관계한 약 80KB의 정보를 포함하는

행렬이다.

```
for(j=0;j<N;j++)                        /*DP단계*/
   for (k=0;k<32;k++)
      fft(data[.][j][k]);

ht(data[1][.][.],house);                        /*HT단계*/
for(i=0;i<N;j++)                        /*BF단계*/
   bf(data[i][.][.],house,detect[i][.]);

for(j=0;j<N;j++)                        /*TD단계*/
   for(i=0;i<N;i++)
         td(detect[i][j],target_report);
```

3. 1. 3. 업무 및 TPC성능평가기준

상업적응용프로그람들에 적용되는 가장 일반적인 성능평가기준들은 Transaction Processing Performance Council(TPC)에 의하여 개발된 TPC성능평가기준이다. 그것은 트랜잭션처리와 자료기지성능평가기준을 개발하기 위하여 나온 단체이다. TPC는 그 성능평가기준에 대한 열린 표준규정을 제공하는데 이것을 임의의 검사자가 실현할수 있고 TPC의 검사자들은 출판전에 결과를 자세히 관찰한다.

지금까지 TPC는 네가지 성능평가기준을 내놓았다. TPC-A와 TPC-B는 1995년 6월 현재로 쓰고 있다. TPC-C는 트랜잭션처리체계의 성능과 가격 대 성능비를 측정하는 자료입력성능평가기준이며 TPC-D는 결심채택지원체계의 성능들을 측정한다.

TPC는 TPC-E(Enterprise)라고 부르는 새로운 성능평가기준에서 대규모업무회사들에 적합한 계산환경을 지원하는 주어 진 체계의 능력을 량적으로 평가하기 위하여 리용되고 있다.

현재 TPC-C는 가장 널리 사용된 상업적응용프로그람성능평가기준이다. TPC는 직결트랜잭션처리(OLTP)성능평가기준이다. 그것은 말단조종자들이 자료기지에 대한 트랜잭션을 실행하는 완전한 소매회사환경을 모의한다. 회사들은 N개의 창고들을 운영하며 매 창고는 10개의 판매지역에 공급하고 매개 지역은 3000명의 구매자들에게 봉사한다. 매 창고는 10개의 말단을 가지며 매 지역마다 하나씩 대응된다.

임의의 시각에 순서를 만들어 구매자의 자료기지에 지불하고 순서상태와 배달순서, 현재 재고준위를 시험하기 위하여 조작자는 표 3-6에서 보여 준 다섯개의 트랜잭션가운데서 하나를 선택한다.

그러나 TPC는 모든 트랜잭션들이 실행될것을 요구하며 최소한 43%가 지불되여야 하고 4%는 매개가 순서상태, 배달과 재고준위여야 한다.

TPC-C결과들을 발표하기 위하여 검사자는 TPC-C규정에서 언급된 모든 요구들이 달성되는가를 보여 주는 충분한 보고를 제공해야 한다. 보고는 구체적인 체계구성정보, 성능, 비용척도들을 제시한다. 전체 체계비용은 5년간 유지비용과 180일동안 운영하는 동

안에 필요한 직결자료보관을 포함하여 모든 하드웨어와 쏘프트웨어를 포괄한다.

표 3-6 TPC-C평가기준에서 다섯가지 형태의 트랜잭션

트랜잭션형태	자료기지접근	트랜잭션무게	실행빈도	90%응답시간
New-Order	읽기/쓰기	중간	높다	<5s
지불	읽기/쓰기	가볍다	적어도 43%	<5s
순서상태	읽기	중간	적어도 4%	<5s
배달	읽기/쓰기	무겁다	적어도 4%	<5s
재고준위	읽기	무겁다	적어도 4%	<20s

두개의 TPC-C결과들이 자주 리용된다. 즉 성능결과 *tpm*C와 비용/성능결과 *$/tpm*C. TPC-C처리량 혹은 분당 TPC-C트랜잭션들은(tpmC로 표현) 분당 처리될수 있는 New-Order의 수로 측정한다. 한편 체계는 TPC-C성능평가기준규정에 의하여 결정된 작업부하혼합에 따라 다른 4가지 형태의 트랜잭션을 실행하고 있다. 비용/성능은 전체체계비용을 나눈것으로 정의한다. 구체적인 실례로 IBM PC봉사기 704는 약 58만 8천딸라의 전체 체계비용, TPC처리량 6679.50tpmC와 비용성능 88$/tpmC를 가진다.

TPC-C는 측정되는 체계가 확대축소되게 하지만 말단의 수와 자료기지의 크기가 비례하여 확대축소되여야 한다. 창고가 회사의 전체 재고를 유지할수 없으므로 다른 창고에 가야 한다. 이것은 TPC-C가 혼란되는 병렬성능평가기준으로 되지 않게 한다. 측정체계는 ACID속성 즉 원자성, 일치성, 고립성, 견딜성을 제공한다.

3. 1. 4. SPEC성능평가기준계렬

SPEC성능평가기준계렬은 Standard Performance Evaluation Corporation에 의해 개발되였다. SPEC는 작은 합성형핵심부대신에 실제적인 작업부하를 잘 반영하는 실제적응용프로그람을 개발하는데 중점을 두고 있다. 매개 성능평가기준과 성능평가기준묶음에 대하여 SPEC는 전체 체계의 종합적인 성능을 측정하는 몇개의(많은 경우 두개) 척도들을 포함한다.

SPEC는 CPU성능을 측정하는것으로부터 시작하였지만 의뢰기/봉사기계산, 상업적응용프로그람 I/O부분체계, 기타 등등에까지 확장되였다. 현재 SPEC는 다음과 같은 성능평가기준묶음들을 내놓았다.

- **SPEC95** 그것은 CPU성능, 기억기체계와 콤파일러코드발생의 성능을 측정한다.
- **SPEChpc96** 이것은 공업적인 응용프로그람들을 실행하는 고성능계산체계의 성능을 측정한다. 그것은 현재 두개의 성능평가기준들을 포함한다. 즉 지진처리조종시험 SPECseis96과 계산화성능평가기준 SPECchem96를 포함한다.
- **SPECweb96** 이것은 실지 세계적인 WWW봉사기의 동작일지로부터 나온 작업부하에 토대한 web봉사기성능을 측정한다.
- **SFS** 체계준위파일봉사를 위하여 그것은 LADDID성능평가기준을 포함하며 여러가지 작업준위에서 NFS파일봉사기의 처리량에 대한 응답시간을 측정한다.

- **SDM** 체계개발다중과제기능을 수행하기 위하여 전형적인 Unix쏘프트웨어개발명령(실례로 make,cp,grep,spell)들을 쓰는 많은 사용자를 가진 환경을 어떻게 운영해 나가는가를 측정한다. 그것은 Unix 쉘스크립으로 만들어 진 두개의 다중사용자 Unix명령을 포함한다.
- **GPC** 도형성능을 특징 짓기 위하여 도형성능을 측정한다. 그것은 진행중에 있는 세개의 개발계획을 포함한다. 그림준위성능평가기준(PLB)은 초당 벡토르와 초당 다각형, 도형성능을 측정한다. X성능특성묘사(XPC)개발계획은 Xmark93성능평가기준도구를 X윈도우즈성능을 측정하기 위하여 개발하였다. OpenGL성능을 측정하기 위하여 Viewperf성능평가기준을 개발하였다.

SPEC95 SPEC95 CPU성능평가기준은 판매자와 사용자들에 의하여 널리 사용되는 가장 유명한 SPEC성능평가기준으로서 이것들은 CPU속도, 캐쉬/기억기체계 그리고 전체로서 콤파일러를 측정하는 CPU성능평가기준이다.

조작체계와 I/O기능에서 소비되는 시간을 무시할수 있다. SPEC95는 CNT95, 8개의 옹근수프로그람모임, 10개의 류동소수점프로그람모임으로 이루어 져 있고 그것들은 둘다 CPU집약적인 응용프로그람들이다.

SPEC는 정기적으로 고정복사와 web를 통해 여러가지 기계의 성능들을 발표한다. 표 3-7은 8MB의 캐쉬와 128MB의 기억기를 가지는 500MHz Alpha 2116처리기를 사용하는 3차원 Alpha Station의 SPEC95결과들을 보여 준다. 주어 진 체계의 모든 SPEC95결과들은 참조기계인 Sun SPARC Station 10/40와 비교되는 비률로서 표현된다. 실례로 값 5는 측정된 기계가 SPARC Station 10/40보다 5배의 성능을 가지거나 또는 4배나 더 빠르다는것을 의미한다.

매 척도는 개별적성능평가기준의 비률을 기하평균하여 전체 성능평가기준묶음으로 집합된다. 속도는 성능평가기준단일복사를 실행하는 비률로 측정하며 한편 구간내 자료량(비률)척도는 성능평가기준의 다중복사를 실행하는 비률로 측정한다. 결과는 피동적인 최량화 혹은 능동적인 최량화로부터 얻어 지며 기준선과 최대성능수들로서 평가한다.

최대속도는(표 3-7에서 굵게 표시한것) 흔히 SPEC95수들을 인용한다.

표 3-7 Alphastation의 SPEC95성능(8MB 외부고속완충기억기와 128MB기억기를 가진 500MHz 21164처리기)

척 도	속 도		처 리 량	
	95	_base95	_rate95	_rate_base95
SPECint	15	12.6	135	113
SPECfp	20.4	18.3	183	165

실례 3.2. SPECint와 SPECfp결과의 해석

Digital Alpha Station 500/5001은 SPECint_rate base95=113과 SPECfp95=20.4를 가진다. 이 수들은 무엇을 의미하는가? SPECint_rate_base95수는 CINT95조의 8개의 성능평가기준의 비률을 기하평균하여 얻어 지는데 매 성능평가기준은 낮은 최량화(실례로_0)로 콤파

일러된다.

매 성능평가기준의 비률은 한주일동안에 성능평가기준의 다중복사를 실행하여 측정되며 실행시간은 Sun SDARC station 10/40에 관하여 정규화된다. 수자 113은 한주일동안에 SunSPARCstation보다 CINT95의 복사를 112배 더 많이 실행한다는것을 의미한다.

SPECfp95수는 CFD95조의 10개 성능평가기준비률을 기하평균하여 얻으며 매 성능평가기준은 능동적인 최량화(실례로 ?O・)로 콤파일러된다. 이 성능평가기준은 Sun SDAR Station 10/14에 관하여 정규화된 실행시간을 가지는 파일복사를 실행하여 측정한다. 수 20.4는 수자기계가 Sun SDAR Station 10/40보다 CFP95의 단일복사실행에 대하여 19.4배 더 빠르다는것을 의미한다.

3. 2. 가격 대 성능비

어떻게 콤퓨터체계의 성능을 재는가? 많은 사람들은 실행시간이 콤퓨터의 성능을 재는 믿음직한 척도라고 믿고 있다. 이 방법은 목표기계에서 사용자의 응용프로그람을 동작시키는것이며 경과된 시간을 측정하는것이다. 직관적으로 이 방법은 때때로 적용하기 어렵고 또한 틀린 해석을 허용할수 있다.

실례로 제한된 응용프로그람들이 각이한 콤퓨터체계에로 이식될수 있다. 새롭게 도입된 병렬콤퓨터에서 주어 진 응용프로그람을 실행하는것은 프로그람작성자들에게 지불하는 높은 비용과 CPU시간은 말할것도 없고 응용프로그람코드들을 변경하고 검열수정하며 검사하는것인데 많은 로력과 시일이 요구된다. 현재 병렬콤퓨터들에서는 CPU시간에 수십부터 수백딸라를 부담하게 된다. CPU시간마다 20딸라라고 하면 응용프로그람이 한 시간동안에 256처리기우에서 실행하는데 5000딸라이상이 부담된다. 병렬코드개발에서 로력비용에 비하면 CPU시간 비용은 비싼것 같이 보이지만 더 작다. 왜냐하면 많은 사용자들이 병렬성콤파일러와 같은 자동화된 도구들을 특히 최대성능이 중대하지 않거나 응용프로그람이 최종판본이 아니고 집합된것일 때 선택하기때문이다.

실례 3.3. 성능측정으로서 실행시간을 리용하는데서 주의할 점

어려운 작업에 대하여 많은 로력을 소비하고 시일이 지난후에 사용자는 최종적으로 코드를 병렬콤퓨터 *X*우에서 성과적으로 실행하고 1000s의 실행시간을 측정했다고 가정하자. 이 실행시간측정은 단독으로는 많다고 볼수 없다.

500s의 실행시간으로 또 다른 기계 *Y*에서 코드실행된다고 가정하면 기계 *X*는 *Y*보다 더 느리다고 결론 지을수 있는가? 대답은 "반드시는 아니다."일것이다. 그것은 기계 *X*는 더 느릴지도 모른다. 그러나 그것은 알고리듬의 특별한 체계에 대하여 최량으로 실현되지 않을수 있거나 나쁜 자료구조가 기계 *X*코드에서 사용될수 있기때문이다.

*X*는 실제적으로 우월한 계산능력을 가지고 있지만 리용자코드는 너무 fine-grain로 되여 과도한 통신부가처리를 일으킨다. 문제는 실행시간 하나로서는 사용자에게 기계의 진짜 성능에 대한 좋은 실마리를 줄수 없다는것이다. 실행시간은 유일한 성능평가로 될수 없다. 사용자들속에서 자주 주장하는 여섯가지 형태의 성능요구 즉 실행시간, 속도, 구간내 자료

량, 리용성, 비용효과성, 성능/비용비률을 아래에서 서술한다. 평가기준들은 같은 콤퓨터가 동환경우에 있는 같은 응용프로그람에 대하여서는 완전히 다른 요구를 가질수 있다.

3. 2. 1. 실행시간과 처리량

표 3-8은 STAP성능평가기준들에 의하여 세개의 프로그람이 256개마디 SP2우에서 실행될 때 여러개의 성능값들을 렬거한다.

표 3-8　　256-마디 SP2에서 측정된 STAP성능

프로그람	실행시간	속도	속도증가	리용
APT	0.16s	9 Gflop/s	90	13%
HO-PD	0.56s	23 Gflop/s	233	34%
GEN	1.40s	3.8 Gflop/s	86	6%

실행시간 여러 응용프로그람들에서 극히 중요하다. 실례로 어떤 실시간응용프로그람에서 리용자는 일감이 시간한계내에 끝나기가 담보되는가에 대하여 관심을 가진다. STAP성능평가기준이 0.5s이내에서 실행되여야 한다고 가정하자. 표 3-6에서 측정된 성능자료들은 오직 APT프로그람만이 256개마디 SP2우에서 이 0.5s요구를 만족한다는것을 보여 준다. HO-PD와 GEN은 이 요구에 이르지 못한다.

처리속도 많은 응용프로그람들에서 사용자들은 실행시간보다 오히려 일정한 처리속도를 얻는데 관심을 두게 된다. 응용프로그람은 각이한 작업부하를 가지고 각이한 자료입력을 처리하며 따라서 각이한 실행시간을 가진다. 그러나 속도요구는 보존되여야 한다.

실례로 STAP프로그람은 성능평가기준이며 거기서 그것들은 실제적인 생산물적인 프로그람은 아니다. 오히려 그것들은 실제코드의 특성을 알아 낸다. 사용자가 바라는것은 10Gflop/s라고 하는 일정한 처리속도를 낼수 있는 하나의 STAD체계이다. 표 3-8로부터 256개마디 SP 2우에서 오직 HO-PD프로그람만이 이 속도요구에 부합된다는것을 알수 있다.

체계처리량 또 하나의 속도와 관련된 요구는 처리량이며 처리량은 단위시간동안에 처리되는 일감의 수로 정의된다. 한 시각에 하나의 일감만이 처리된다. 처리량은 실행시간과 거꿀이다. 실례로 표 3-8에서 256개 마디 SP 2의 APT프로그람에 대한 처리량은 0.16s마다 하나의 APT이며 초당 6.25APT이다.

처리량은 보통 다중일감이 동시에 실행될 때 사용된다. 많은 경우들에서 체계처리량은 다음의 두 방법에 의하여 증가시킬수 있다.

- 하나의 방법은 관흐름화에 의한 방법이다. 여기서 련속인 일감들은 여러 관흐름단계에서 그 실행을 중첩한다. 처리량은 가장 긴 관흐름선단계실행시간에서 거꿀로 된다.
- 다른 방법은 매개 마디에 개별적일감을 배정하는 방법이다. *n*개의 일감은 *n*개마

디에서 동시에 처리된다.

실례 3.4. 병렬 APT에서 처리량과 속도의 계산

표 3-8로부터 APT프로그람의 작업부하가 9*0.1b=1.44Gflop라는것을 유도할수 있다. 단계마다 128개 마디들을 사용하여 256개 마디 SP 2에서 APT프로그람에 대한 두 단계관 흐름을 구성할수 있다.

한 단계는 0.14s 걸리고 한편 다른것은 0.11s 걸린다. 따라서 전체 실행시간은 0.16s로부터 0.25s로 증가한다. 그러나 처리량은 초당 1/0.16로부터 1/0.14=7.14APT로 증가하며 또는 16.34Gflop/s와 등가이다.

APT프로그람은 하나의 SP2마디우에서 실행되는데 약 14s 걸린다. 256마디의 매개에 몇개의 APT들이 달리도록 배정할수 있다. 그다음 체계처리량은 초당 256/14=18.294APT로 되며 또는 25.6Gflop/s와 등가이다.

14s는 실시간탐지기신호처리에 대하여 너무 긴 경과시간이다. 그러나 직결트랜잭션처리와 같은 응용프로그람들에서 받아 들일수 있다. 많은 개발제작환경들에서 그것은 실행시간한계의 모임에 대한 제한밑에서 처리량을 최대화하는것이 공통적인 체험이다.

실례로 사용자는 직결트랜잭션체계에 대하여 1000개의 트랜잭션을 처리하도록 요구를 규정할수 있고 매개 개별적인 트랜잭션에 대한 최대응답시간은 10s보다 많지 않게 규정할수 있다.

3. 2. 2. 사용률과 비용효과성

가장 짧은 실행시간을 찾을 대신에 사용자는 보다 효과적인 비용으로 응용프로그람을 실행할것을 요구할수 있다. 부하불균형, 통신부가처리 등으로 하여 랑비되는것 대신에 CPU-시간의 많은 몫이 쓸모 있는 계산을 위해 리용되기를 바란다. 비용효과성의 훌륭한 지적자는 리용인자(간단히 리용)이며 이것은 주어 진 콤퓨터최대속도에 대한 도달된 속도의 비률이다. 다음실례는 이 용어에 대한 물리적의미를 준다.

실례 3.5. APT프로그람에 대한 IBM SP 2의 리용

매 SP2마디는 266Mflop/s의 최대속도를 가진다. 매 CPU-시간은 10딸라에도 부담된다고 가정하자. 표 3-8로부터 256개 마디우와 1개 마디우에서 APT프로그람실행에 대한 계산을 통해 어느것이 비용효과성이 더 큰가를 보자.

256개 마디 SP 2는 266×250=68Gflop/s의 최대속도를 가진다. APT프로그람은 256개 마디우에서 9Gflop/s에 도달하고 1개 마디우에서 100Mflop/s에 도달하였으며 따라서 각각 13.3%와 37.6%의 리용을 얻는다. 이로부터 한 마디우에서 실행하는것이 보다 더 비용이 효과적이라는것을 보여 준다.

사실상 256개 마디에서 하나의 APT를 한번 실행하는데 0.16s 걸린다. 하나의 APT실행에 대한 전체 비용은 $10×256×0.16/3600=$0.11이다. 대응하는 비용효과성은 딸라당 (9Gflop/s) 10.11$=82Gflop/s이다. 한 마디우에서 실행시간은 약 14.4s이다. 전체 비용은 다만 10$×14.4/3600=0.04$이며 비용효과성은 딸라당 (9Gflop/s)/0.04$=255Gflop/s이다.

이 실레는 높은 리용이 딸라당 높은 Gflop/s에 대응한다는것을 설명해 준다. 이것은 CPU-시간이 고정된 비률로 부담된다면 언제나 옳다.

실행시간속도의 리용은 가장 중요한 척도라는것을 알수 있다.

리용척도에 특별한 주의가 돌려 져야 하는데 그것은 흔히 스쳐 지나가 무관심되고 있지만 실행시간과 속도보다 정보가치가 훨씬 높다. 낮은 리용은 언제나 불충분한 프로그람이나 콤파일러를 가리킨다. 대조적으로 훌륭한 프로그람은 큰 작업부하로 하여 오랜 실행시간을 가질수 있고 기계가 느리면 속도가 낮을수 있다.

그런데 얼마만한 리용값들이 낮다고 고찰되는가 하는것은 물론 응용프로그람과 가동환경에 의존되지만 우리의 경험과 다른 MPP사용자들과의 호상작용은 다음과 같은 평가를 얻는다. 단일 MPP처리기우에서 실행하는 순차응용프로그람은 5%에서 40%까지의 범위에서 리용을 가지며 대체로 8%로부터 25%사이이다. 몇개의 개별적부분프로그람들은 더 빨라서 75% 또는 그 이상에 이르게 할수 있다.

그러나 이러한 부분프로그람들이 실제적인 응용프로그람에 병합되면 그것들은 반드시 높은 리용을 보존하지 못한다. 다중병렬처리기들우에서 실행하는 병렬응용프로그람은 1%에서 35%까지의 범위에서 리용을 가지며 대체로 4%부터 20%사이이다. 일반적인 그릇된 견해는 병렬계산의 외부통신과 무익한 부과처리를 가질 때 단일마디 또는 순차계산은 언제나 높은 리용을 가진다는것인데 이것은 언제나 옳지 않다. 실례로 Intel paragon에서 실행하는 병렬 APT프로그람은 한개 마디가 아니라 네개 마디우에서 가장 높은 리용을 가지였다.

NA와 STAP성능평가기준을 사용하여 세개의 MPP(Intel Paragon, IBM SP2, Cray T3D)

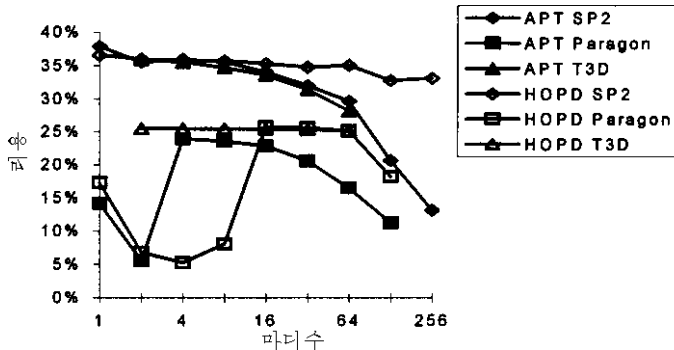



그림 3-1. 세개의 MPPs에서 병렬 APT와 HO-PD성능평가기준프로그람의 리용

표 3-9 6개 콤퓨터의 NAS평가기준리용

형태	기계모형	최대	최소	조화평균
PVP	Cray C90	73	28	54
MPP	Cray T3D	49	2	10
	IBM SP2	60	3	18
	Intel Paragon	36	3	9
SMP	SGI Power Challenge	44	10	25
	Convex Exemplar	35	1	8

우에서 측정된 리용값들을 아래에서 고찰한다. 그림 3-1에서는 두개 병렬STAP프로그람의 사용률과 속도를 측정한데 대하여 보여 준다.

리용은 5%부터 38%의 범위이다. 일반적으로 리용은 더 많은 마디들이 사용될 때 떨어 진다. 그러나 이것은 언제나 옳지 않다. NAS병렬성능평가기준결과[525]들을 처리하여 표 3-9와 그림 3-2에서 보여 준다.

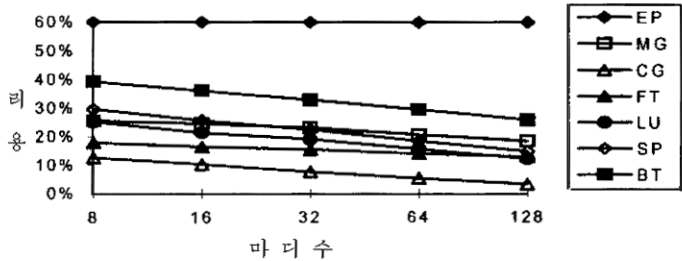
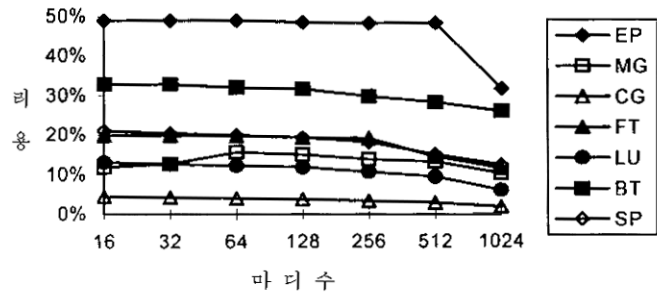
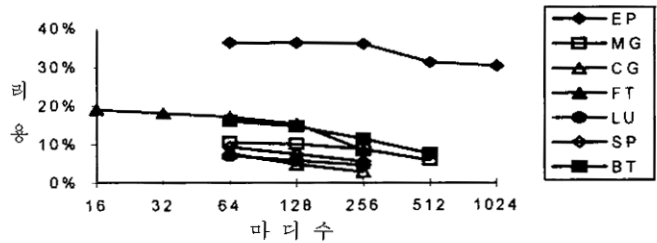



그림 3-2. NAS병렬평가기준에 대한 세개의 MPPs의 리용

세개의 MPP들에서 리용은 2%로부터 60%까지의 범위에 있으며 조화평균은 12%이다. 이 결과들은 판매자성능평가기준프로그람으로부터 얻어 졌는데 고도로 최량화되였다. 류사한 형태를 가지는 사용자프로그람은 같은 성능을 얻는데서는 류사하지 않다.

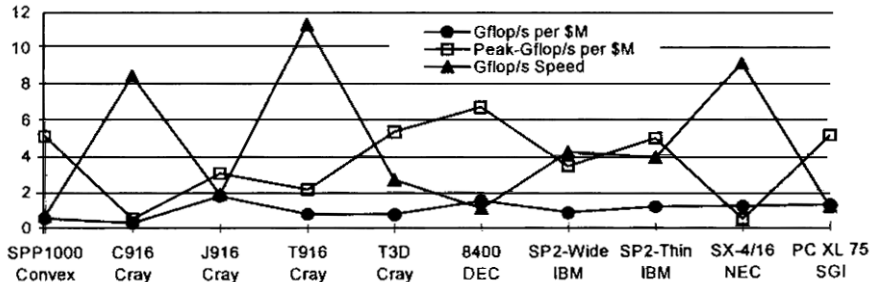



그림 3-3. 10개의 1995년 병렬콤퓨터의 성능/비용

비용효과성척도는 콤퓨터체계의 성능이 비용률에 용해되지 않으며 그것은 구입비용에 대한 속도의 비률로 정의된다. 비률은 NAS성능평가기준을 실행하는 다양한 콤퓨터들에 대하여 그림 3-3에서 설명한다.

백만딸라당 최대 Fflop/s로 측정되는 성능/비용률은 넓은 변동에 있지만 1995년에 지속적인 성능/비용률은 백만딸라당 1Gflop/s근방에 훨씬 더 많이 집중되였다.

실례 3.6. 잘못 평가된 최대성능/비용률

체계를 비교하기 위하여 최대성능/비용률을 리용하는것은 흔히 잘못될수 있다. 그림 3-3에서 Cray J916의 최대성능/비용률은 Convex SPP1000, Cray T3D, SGI Power Challege보다 훨씬 더 낮다. 그러나 그것의 지속적인 성능/비용률은 실제적으로 더 높다.

흔히 사용자들은 서로 다른 체계들을 비교하는데서 하나이상의 척도를 사용할 필요가 있다. 비용효과성이나 성능/비용률을 하나만 리용하면 현재의 IBM PC는 대형콤퓨터나 고속콤퓨터에 비해 PC의 비용이 훨씬 더 낮기때문에 보다 강력한 체계들을 쉽게 릉가할수 있다.

개괄하면 실행시간은 리용자가 요구하는 여러 성능들중의 하나이다. 다른것들은 속도, 처리량, 리용, 비용효과성, 성능/비용률, 확대가능성이다. 보통 요구모임이 제기된다. 실례로 리용자는 일정한 준위의 처리량과 리용을 실행시간제한밑에서 요구하게 된다. 사용자는 최소속도에서 가장 좋은 성능/비용률을 가지는 체계를 선택하려고 한다.

3. 3. 기본성능척도

여기서는 병렬콤퓨터들과 마찬가지로 단일처리기체계에서 리용된 성능척도들을 연구한다. 용어는 Parkbench그룹에 의해서 제안된것과 일치하며 물리와 같은 다른 과학분야들에서 사용되는 용어를 그대로 따른다.

3. 3. 1. 작업부하와 속도척도

프로그람 C의 계산작업부하를 재는데 빈번히 리용되는 세가지 척도를 보면 실행시간, 실행된 명령수, 실행된 류동소수점연산수이다. 첫번째 척도는 특정한 콤퓨터체계에 구속된다. 그것은 프로그람 C가 다른 기계우에서 실행될 때 변화될수 있기때문이다.

표 3-10 작업부하와 속도척도

작업부하형태	작업부하단위	속도단위
실행시간	초 (s), CPU박자	초당응용프로그람
명령계수값	백만개 혹은 10억개명령	MIPS 혹은 BIPS
류동소수점연산(flop) 계수값	Flop 백만 flop(Mflop) 10억 flop(Gflop)	Mflop/s Gflop/s

두번째 척도는 명령모임구조(ISA)에 속박되며 프로그람이 같은 ISA를 가지는 다른 기계우에서 실행될 때 변화되지 않는다. 세번째 척도는 흔히 구조적으로 독립이다. 표 3‒10은 세가지 성능척도를 모두 개괄하였다.

명령계수값 임의의 프로그람 *C*에 대하여 작업부하의 척도로서 명령계수값을 사용할수 있는데 그것은 실행되는 명령의 수이다. 이러한 작업부하는 백만단위의 명령들을 가진다. 대응하는 속도척도는 초당 명령의 백만단위 MIPS이다. MIPS비률을 리용하는데 다음과 같은 몇가지를 주의해야 한다.

- 작업부하는 아쎔블러프로그람본문(정적프로그람계수값)에서 명령수가 아니라 기계가 실행한 명령들이다(즉 동적명령계수값).
- 명령계수값은 입력자료값에 관계된다. 실례로 정돈프로그람은 주어 진 입력에 대하여 100000개 명령들을 수행할수 있지만 다른것에 대해서는 2000개만을 실행할수 있다. 이러한 입력의존적인 프로그람에 대하여 작업부하는 나쁜 경우의 입력 또는 특수한 기준선입력자료모임에 대한 명령계수값으로 정의된다.
- 입력을 고정하여도 실행된 명령은 각이한 기계들에서 같지 않다. 실례로 RISC처리기는 동일한 고준위의 언어가 주어 졌을 때 CISC처리기보다 50%부터 150%나 더 많은 명령들을 실행한다.
- 고정된 입력에 대하여 각이한 콤파일러나 최량화선택이 리용될 때 동일한 기계우에서 프로그람의 명령계수값은 차이날수 있다.
- 마지막으로 명령계수값이 크다는것은 프로그람이 더 많은 실행시간을 요구한다는것을 꼭 의미하지 않는다.

실행시간 특정한 콤퓨터체계우의 어떤 주어 진 프로그람에 대해서 작업부하를 프로그람을 실행하는데 걸리는 전체 시간으로 정의할수 있다. 이 실행시간은 벽시계시간으로 측정해야 한다. 그 시간은 경과시간으로 알려 져 있는데 실례로 Unix함수 gettimeofday()에 의해 구할수 있다. 작업부하의 기본단위는 초이며 분, 시간, 미리초(ms), 마이크로초(μs)도 쓸수 있다.

실행시간은 많은 인자들에 의존되는데 몇가지 중요한것들을 아래에서 고찰한 다.

- **알고리듬** 사용된 알고리듬은 실행시간에 큰 영향을 준다. 실례로 N개 항목을 정돈하기 위한 정돈알고리듬은 $O(N^2)$의 실행시간을 가지며 병합정렬은 다만 $O(\log N)$을 가진다.
- **자료구조** 자료를 어떻게 구조화하는가 하는것도 성능에 영향을 준다.
- **입력자료** 많은 응용프로그람의 실행시간은 입력자료값에 의존하지 않는다. 실례로 N점 FFT프로그람은 $O(N\log N)$시간을 가지며 벡토르가 무엇인가에는 관계 없다. 정렬과 탐색프로그람과 같은 다른 프로그람들은 각이한 입력자료에 대해 각이한 시간을 취할수 있다. 왜냐하면 프로그람조종흐름이 각이한 입력자료에 따라 변하기때문이다. 그러한 입력의존성프로그람에 대해 실행시간을 작업부하로

리용할 때 나쁜 경우의 시간 혹은 기준선입력자료모임을 쓸 필요가 있다.

- **가동환경** 명백히 기계하드웨어와 조작체계는 성능에 영향을 준다. 그렇게 명백치 않은것은 성능이 처리기가 아닌 다른것의 영향을 받는다는것이다. 동일한 박자률에서 동일한 처리기를 사용하는 두 가동환경우에서 완전히 다른 시간결과를 볼수 있다. 다른 인자들로는 계층기억기(캐쉬, 주기억기, 디스크), 조작체계변종 그리고 응용프로그람이 콤퓨터의 사용을 전용화하는가 또는 다른 응용프로그람들과 자원을 시분할하는가 하는것이 될수 있다.
- **언어** 동일한 알고리듬과 자료구조가 동일한 가동환경에서 리용될 때 응용프로그람을 코드화하는데 사용되는 언어들이 서로 다르다면 각이한 실행시간이 걸린다. 게다가 사용된 콤파일러와 콤파일러/련결편집선택은 중요한 역할을 논다. 2진준위서고함수들을 사용하는것 역시 실행시간을 줄인다.

류동소수점계수값 수값계산을 위주로 하는 과학, 공학계산과 신호처리응용프로그람에 대하여 본질적인 척도는 실행되는데 필요한 류동소수점연산수이다. 표 3-11은 실천에서 리용된 몇개의 규칙들을 설명한다.

응용프로그람 *C*가 간단하고 그것의 작업부하가 입력에 의존하지 않을 때 *C*의 작업부하는 코드를 검사하여 결정할수 있다. 응용코드가 복잡하거나 작업부하가 각이한 입력자료에 따라 달라 질 때(실례로 정돈과 탐색문제들) 특정한 기계우에서 응용프로그람을 실행하여 작업부하를 측정할수 있다.

표 3-11 **류동소수점연산을 계수하는 규칙**

연산	Flop 계수	규칙설명
A[2*i]=B[j-1]+1.5*C-2	3	더하기, 덜기 혹은 곱하기를 1 flop로서 계수, 첨수연산은 계수되지 않는다. 배정은 개별적으로 계수되지 않는다.
X=Y;	1	고립배정은 1 flop로서 계수된다.
If(X>Y) Max=2.0*X	2	비교는 1 flop로서 계수된다.
X=(float)I+3.0;	2	형태변환은 1 flop로서 계수된다.
X=Y/3.0+sqrt(Z)	9	나누기 혹은 두제곱뿌리는 4 flop로서 계수된다.
X=sin(Y)-exp(Z)	17	시누스함수, 지수함수는 8 flop로서 계수된다.

이 특수한 실행은 flop의 수 또는 실제적으로 실행된 명령의 수에 대한 결과를 얻어낸다. 이 방법은 NAS성능평가기준에서 사용되였으며 거기서 flop계수값은 Cray Y-MD 또는 Cray C90에서 실행되여 결정된다.

실례 3.7. FFT처리에서 리용률

실례 3.10을 다시 고찰하자. 매 *N*점 FFT는 작업부하 W=5Nlog$N$flop를 가진다.

콤퓨터처리단계에 대한 전체 작업부하는 2048×(5×8192×log8192)flop이거나 또는 약

1.09Gflop이다.

기계 X우에서 50s의 실행시간을 가지면 속도는 약 22Mflop/s이다. 기계 X가 266Mflop/s의 속도를 가진다고 하자. 도플러처리는 리용률 22/266=8.27%를 가지는데 최대성능에 비해 오히려 낮은 률이다.

리론적인 성능해석에서 자주 명령이나 flop는 동일한 시간량을 가진다고 가정한다. 이 유일속도과정은 실지체계에서는 성립하지 않는다. 실례로 단일 IBM SP2마디우에서 속도는 5부터 250Mflop/s까지 변한다는것을 볼수 있다. 따라서 순차실행시간은 또한 작업부하를 측정하는 명령계수값이나 flop를 보충완비하는데서 널리 쓰인다.

3. 3. 2. 순차성능에 대한 추가적설명

작업부하척도로 실행시간이나 명령계수값을 리용하면 이상한 현상에 부닥친다. 작업부하는 같은 응용프로그람이 다른 체계에서 실행될 때 변한다. 게다가 작업부하는 오직 프로그람을 실행하여 결정할수 있다.

Flop계수값척도는 보다 안정하다. 실례로 FFT는 그것이 어떻게 실행되는가에 관계없이 동일한 작업부하 W=5Nlog$N$flop를 가진다. 보충적으로 flop계수값(더 낮은 정도로, 명령계수값)은 실행하지 않고 코드를 검사하여 결정할수 있다.

Flop계수값과 명령계수값은 다른 우점을 가진다. 그것들의 대응하는 속도와 리용측정들은 응용프로그람이 어떻게 잘 실행되는가 하는 몇가지 평가기준을 준다.

현재 고급언어프로그람들은 흔히 순차콤퓨터우에서 5%에서 40%까지의 리용에 도달하고 있으며 병렬콤퓨터우에서는 1%에서 25%까지이다.

높은 수준으로 작성된 프로그람들은 더 높이 도달할수 있다. 실례로 Agarwal 등[13]은 POWER2처리기의 특수한 구조적특징들을 리용하기 위하여 포트란프로그람들을 작성하면 몇가지 수값부분프로그람들에서 90%의 리용에 도달할수 있다.

실례3.8. IBM SP2의 순차성능

표 3-12는 단일 SP2마디우에서 STAP성능평가기준프로그람들의 순차성능을 보여 주며 그것은 266-Mflop/s의 최대속도를 가진다.

작업부하값은 원천 STAP프로그람을 검사하여 얻는다. 실행시간값은 매 구성요소알고리듬들을 실제적으로 측정한데로부터 나온다.

실례로 APT프로그람은 네개의 구성요소알고리듬으로 갈라 진다. 즉 : 도플러처리(DP), Householder Transform(HT), Beamforming(BF), Target Detection(TD)로 갈라 진다.

매 구성요소알고리듬은 각이한 실행시간을 가지는 각이한 량의 작업부하를 수행하며 따라서 각이한 속도와 리용값을 얻는다. 표 3-12에서 리용값들로부터 곧 SP2의 POWER2구조에 대하여 FFT프로그람은 너무 느리여 락관적이지 못하다는것을 알수 있다. 다른 한편 beamforming프로그람은 아주 좋으며 이러한 내용들을 실행시간만으로는 찾아 볼수 없다.

성능척도에 대한 개괄 종합해 보면 세가지 척도들은 모두 쓸모가 있다. 병렬기계에

서 외부연산들이 같은 문제를 풀기 위하여 실행될 필요는 있지만 응용프로그람성능을 예측하고 측정하는데서 중요한것은 단일작업부하를 무모순으로 사용한다는것이다.

표 3-12 SP2에서 순차 STAP프로그람성능

프로그람	구성알고리듬	작업부하 (Mflop)	실행시간 (s)	속도 (Mflop/s)	리용 (%)
APT	Total	1447	14.37	100	37.85
	DP	84	4.12	20	7.65
	HT	2.88	0.04	72	27.07
	BF	1314	9.64	136	51.22
	TD	46	0.57	75	28.01
HO-PD	Total	12,853	130.61	98	37.00
	DP	220	11.62	19	7.12
	BF	12,618	118.82	106	39.92
	TD	14	0.17	82	30.96
GEN	Total	5326	121.05	44	16.54
	DP	1183	22.80	52	19.51
	HT	1909	79.14	24	9.06
	BF	604	19.11	32	11.88
	TD	1630	20.23	82	31.00

실천적으로 flop계수값작업부하는 흔히 표 3-11의 규칙들에 따라서 코드를 검사하여 결정한다. 실행시간은 특정한 기계(실례로 SPARCstation 또는 Cray Y-MP)에서 하드웨어, 가동환경, 콤파일러선택, 입력자료모임 등으로 이루어 지는 특정한 검사조건에 기초하여 측정된다.

3．4．병렬콤퓨터의 성능

이 부분에서는 병렬콤퓨터체계의 성능속성들을 론의하며 다음부분에서 병렬응용프로그람의 성능속성들에 대하여 론의한다. 이미 1.3.2에서 언급한것처럼 병렬체계는 계산적이고 그리고 부가처리의 특성들을 가진다.

3．4．1．계산특성

표 3-13은 세가지 상업적병렬콤퓨터계렬에 대하여 성능파라메터들에 의한 리력적인 값들을 보여 주었다. 여기서 Cray계렬은 PVP를, Intel계렬은 MPP을, SGI계렬은 SMP를 표현한다.

계층기억기 현대 콤퓨터의 성능은 체계가 얼마나 빨리 처리기들과 기억들사이에서 자료를 움직일수 있는가에 의존한다. 기억기부분체계계층이 그림 3-4에 제시된다. 매층에

대하여 세가지 파라메터들을 고찰한다.

- **용량 C** 얼마나 많은 byte자료를 장치가 관리할수 있는가?
- **지연시간 L** 장치로부터 한 단어를 꺼내는데 얼마만한 시간이 필요한가?
- **대역너비 B** 대량자료를 움직일 때 1s에 장치로부터 얼마만한 byte가 전송될수 있는가?

표 3-13 3개 콤퓨터계렬의 계산파라메터

콤퓨터	년도	박자률 (MHz)	기억기 용량	기계크기 n	P_{peak} (Mflop/s)	n・P_{peak} (Gflop/s)
Cray 1	1979	80	1MB	1	160	0.16
Cray X-MP	1983	105	4MB	2	210	0.42
Cray Y-MP	1987	166	1GB	8	333	2.66
Cray C90	1991	238	2GB	16	1000	16
Cray T90	1995	454	8GB	32	2000	60
Intel iPSC	1985	10	64MB	128	0.04	0.005
Intel iPSC/8601992	1989	40	1GB	128	60	7.68
Intel Paragon	1992	50	64GB	2048	75	153
Intel ASCI	1996	200	297GB	9216	200	1800
SGI Power	1988	16	256MB	2	1	0.002
SGI Challenge	1993	150	16GB	36	75	2.7
SGI Power Challenge XL	1994	75	16GB	18	300	5.4

현재의 콤퓨터들에서 세 파라메터의 전형적인 값들이 그림 3-4에 보여 주었다. 장치가 빠르고 작을수록 처리기와 더 가깝다. 처리기에 가장 가까운 장치는 등록기인데 그것은 사실상 처리기소편의 한 부분이다. 일반적으로 등록기들로부터 기능단(실례로 ALU)에로 한 단어를 꺼내는데 외부주기들은 필요하지 않다. 따라서 지연시간은 0이다. 그러나 등록기들은 아주 제한된 용량을 가진다. 실례로 Intel 80x86처리기들은 32byte를 유지할수 있는 8개의 등록기를 가진다.

다른 처리기들(실례로 RISC처리기들과 벡토르처리기들)은 보다 많은 등록기들을 가지지만 그 용량은 2KB를 초과하지 않는다. 대역너비는 다음의 방법으로 평가할수 있다.

더하기연산은 등록기로부터 두 단어를 꺼낼 때 한 단어를 등록기에 보관한다. 주기억은 100MHz이고 단어길이가 64bit라고 가정하자.

대역너비는 $3 \times 8 \times 100 \times 10^6 = 2.44B/s$이다. 주기률이 더 빠르고 처리기안에서 병렬연산을 실행하므로 대역너비는 더 높아 진다.

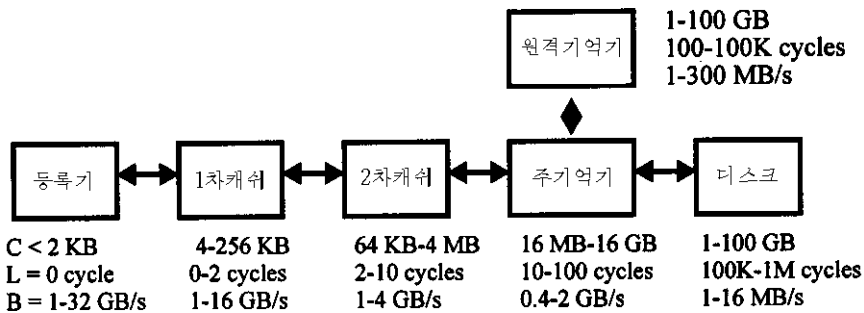



그림 3-4. 대표적인 계층기억기의 성능파라메터

1차캐쉬는 보통 처리기소편우에 있다. 2차캐쉬는 소편과 떨어 져 있다. 주기억기는 마디안에 있는 국부기억기와 PVP와 SMP와 같은 집중형공유기억기를 가진 기계에 대한 대역기억기를 포함한다.

원격기억기는 다른 마디들의 모든 국부기억기를 가리킨다.

실례3.9. Digital의 TruCluster기억계층의 성능

DEC TruCluster는 8포구기억기통로호상련결에 의해서 호상접속된 8개의 8400 SMP로 구성된다. 매 SMP마디는 12개까지의 처리기 혹은 14GB까지의 공유주기억기를 가질수 있다. 매 처리기는 16KB 1차캐쉬와 96KB 2차캐쉬를 가지며 둘 다 소편우에 있다.

두 처리기에 의해 공유된 4MB 3차캐쉬는 소편밖에 있다. 다른 마디들의 원격기억기는 기억기통로호상접속을 통하여 접근된다.

표 3-14는 처리기주파수가 300MHz일 때 TruCluster기억계층의 파라메터들을 보여준다. 이것들은 초과될수 없는 가장 좋은 수(하드웨어최대)이다. 사용자처리에 의하여 도달될수 있는 가장 좋은 수들은 괄호안에서 보여 준다. 300MB/s 주기억기대역너비는 하나의 처리기가 기억기에 접근함을 가리킨다. 대역너비는 다중처리기들이 기억기에 동시에 접근할 때 982MB/s만큼 높일수 있다.

표 3-14 TruCluster계층기억기의 성능

장소	용량	대역너비	지연시간
등록기	512B	34GB/s	3ns (1 clock)
1 차캐쉬	16KB	9.6GB/s	3ns (1 clock)
2 차캐쉬	96KB	4.8GB/s	10ns (3 clocks)
3 차캐쉬	4MB	1.2GB/s	20ns (6 clocks)
주기억기	14GB	1.6GB/s [300MB/s]	70ns (21 clocks) [240ns (80 clocks)]
원격기억기	98GB	100MB/s [61MB/s]	5μs (1500 clocks)

3. 4. 2. 병렬성과 호상작용부가처리

리론적인 PRAM모형에서 병렬성과 호상작용부가처리는 무시된다. 실례로 명령준위 동기성은 암시적으로 매 명령다음에 하나의 장벽을 비용이 없이 놓는다.

임의의 기억기단어를 읽고 쓰기하는데서는 시간이 걸리지 않는다. 실제적인 병렬콤퓨터들에서 이러한 가정들은 맞지 않는다. 중첩을 가정하지 않으면 병렬프로그람을 실행하는 시간은

$$T = T_{comp} + T_{par} + R_{interact} \qquad (3.2)$$

이다.

여기서 $T_{comp}, T_{par}, T_{interact}$는 계산과 병렬성, 호상작용조작을 실행하는데 걸리는 시간이다.

1.3.2와 1.3.3에서 병렬프로그람의 부가처리는 세 부류 즉 부하불균형부가처리, 병렬성부가처리, 호상작용부가처리(동기화, 통신, 집합을 포함하는)로 나눈다.

부하불균형은 어떻게 달성하는가 하는것이 흔히 병렬알고리듬설계문제이다. 이 부분에서는 부가처리의 다른 두 부류에 초점을 둔다.

2.3의 론의로부터 알수 있는바와 같이 병렬성연산에는 세가지 형태가 있으며 그것들은 병렬성부가처리의 원인들이다.

- 생성, 종결, 문맥절환 등등과 같은 프로쎄스관리.
- 프로쎄스그룹생성과 취소와 같은 그룹프로쎄스조작
- 프로쎄스식별, 위수그룹식별, 그룹크기를 묻는것과 같은 프로쎄스질문조작

2.4.1에서 본바와 같이 호상작용연산에는 세가지 형태가 있는데 그것은 호상작용부가처리의 원인들이다.

- **동기화** : 장벽, 차단, 림계범위, 사건
- **집합** : 감소와 주사
- **통신** : 점대점과 집합통신 그리고 공유변수의 읽기, 쓰기

이 부가처리를 알면 프로그람작성자가 병렬프로그람을 어떻게 개발하는것이 제일 좋은가를 결정할수 있다. 실례로 병렬성부가처리가 작으면 프로그람작성자는 동적병렬프로그람을 작성하는데 여유를 가질수 있으며 동적병렬프로그람에 의해서 프로쎄스들은 생성되거나 취소되며 문맥절환될것이다.

이 부가처리들이 크면 정적병렬프로그람들이 대신에 리용되여야 한다. 여기서 프로쎄스들은 오직 시작에서 생성되며 제일 끝에서만 종결된다. 장벽들이 기계에서 효과적으로 지원되면 문제를 푸는데서 동기화된 반복알고리듬을 사용할수 있다. 그러나 차단들이 시간을 소비하므로 비동기알고리듬을 사용하지 않는것이 더 좋다.

거대한 부가처리 현대 콤퓨터를 연구할 때 곧 병렬성과 호상작용부가처리에 대한

두가지 문제에 주목한다. 그것은 흔히 기본계산시간과 크게 관계되며 체계마다 취급을 달리 할수 있다. 발생되는 부가처리를 측정하는데서 세가지 척도를 리용한다. 기본척도는 실행시간(μs)이다. 때때로 동일한 량의 시간에서 하나의 마디에 의하여 실행될수 있는 박자주기의 수 혹은 flop수를 리용할수 있다.

실례로 SP2는 66MHz, 266Mflop/s POWER2처리기들을 사용한다. SP2에서 보내기와 받기는 최소한 39s 걸리는데 그것은 2601 CPU박자주기와 동등하거나 또는 10374최대flop와 동등하다. 표 3-15는 McVoy의 Lmbench프로그람에서 얻어 진 몇가지 단일 Unix체계우에서의 프로쎄스관리와 통신의 부가처리를 보여 준다[440].

프로쎄스생성렬은 실패하면 즉시에 탈퇴하는 빈 새끼프로쎄스를 가지치는데 필요한 시간을 보여 준다.

문맥절환렬은 두 프로쎄스사이에서 절환되는데 필요한 시간을 보여 준다. 관지연시간렬은 하나의 관을 통하여 작은 어휘들을 뒤로 보내거나 앞으로 내오는데서 두 Unix처리들에 대한 왕복시간을 보여 준다. 관대역너비는 관을 통하여 두 프로쎄스사이의 50MB의 통보문에 도달될수 있다는것을 보여 준다.

표 3-16은 Solaris조작체계[601]를 실행하는 Sun워크스테이션에서 프로쎄스생성과 동기화에 대한 부가처리를 보여 준다. Solaris는 프로쎄스의 세 형태들 즉 2, 7, 12장에서 론의한바와 같이 무거운 부하 Unix처리, 가벼운 부하처리(LWP), 사용자준위스레드를 지원한다. Unix처리들의 부가처리들은 하나 또는 두 수준 더 높다.

이 자료들은 병렬성과 호상작용부가처리가 아주 클수 있다는것을 보여 준다. 실례로 POWER2처리기는 박자주기당(15ns) 4개의 류동소수점연산을 수행할수 있다.

표 3-15　　Unix체계에서 병렬성과 통신부가처리

처리기	조작체계	프로쎄스생성 (μs)	문맥교환 (μs)	관지연시간 (μs)	관대역너비 (MB/s)
POWER2	AIX 3	1.4K	21	138	N/A
POWER	AIX 2	2.0K	20	143	34
Pentiun	Linux 1.1	3.3K	66	157	13
Alpha	OSFI V2.1	4.8K	25	185	32

표 3-16　　Solaris에서 병렬성과 호상작용부가처리

연　　산	부가처리
Unix프로쎄스가지치기	3,057μs
가볍게 무게화된 프로쎄스생성	422μs
사용자준위스레드생성	101μs
프로쎄스준위차단	105μs
스레드준위차단	1.8μs

표 3-15부터 Unix처리를 생성하는 시간(1.4Kμs)은 372000flops를 실행하는데 충분히

긴 시간이다.

지어 같은 처리기가 사용될 때조차 부가처리값은 역시 한 체계로부터 다른 체계사이에서 크게 달라 진다. 사용자들은 과거의 “류사”한 체계에 대한 경험으로부터 부가처리가 존재하겠는가를 추정할수 없다. 사용자가 비용이 드는 병렬성과 호상작용연산을 쓰지 않도록 하는 부가처리값을 아는것이 중요하다.

흔히 부가처리는 주로 OS핵심부나 체계쏘프트웨어비용에 의하여 생겨 난다. 따라서 부가처리는 체계마다 완전히 다르며 지어 체계들이 같은 처리기구조를 쓸 때조차 다르다.

SP2에서 절환의 하드웨어지연시간은 1μs보다 작다. 그러나 지연시간은 핵심부준위 IP규약을 사용한다면 수백 μs로 대폭 늘어 난다. IBM은 핵심부를 무시하는 사용자공간규약을 실현하는데 그것은 지연시간을 39μs만큼 작게 감소시킨다.

3. 4. 3. 부가처리의 정량화

병렬성과 호상작용부가처리들은 중요하기때문에 정량화하여야 한다. 정량화된 부가처리들은 실제적인 콤퓨터에서 여전히 부족점을 가지고 있다. 병렬콤퓨터회사들은 마디 또는 n개마디체계당 피크성능(MIPS 또는 Mflop/s)과 같은 콤퓨터성능자료들을 여러가지 형식으로 제공한다.

그와는 반대로 점대점통신에서는 시동과 대역너비를 제외하고 부가처리자료들을 거의 제공하지 못한다. 이 문제에 대한 대답은 어렵다. 체계를 생성하고 하나의 프로쎄스그룹를 분할하며 장벽동기화를 진행하고 임의로 현존병렬콤퓨터우에서 집중감시를 수행하는데 얼마나 오랜 시간이 걸리는가 하는 문제는 때때로 판매자들조차도 답변을 하지 못한다.

사용자들은 흔히 자체로 부가처리를 측정해야 하며 오히려 그것은 비용이 드는 문제이다.

병렬콤퓨터들에 대한 수많은 성능평가기준과 척도들이 제기되였지만 부가처리평가는 조금밖에 제공하지 못하였다. 부가처리를 측정하는 현존성능평가기준들은 PARKBENCH 성능평가기준묶음을 포함하는데[317] 그것은 점대점통신과 분산기억 MPP에 대한 장벽동기화를 측정한다.

점대점통신에 대하여[314] Hockney[313]이 제안한 파라메터 r_{∞}, $m_{1/2}$, t_0과 π_0는 잘 알려 진 부가처리척도들이다. 병렬콤퓨터판매자들은 부가처리를 정량화하기 위하여 닫긴 형식의 표현들의 모임을 제공하였다.

왜냐하면 판매자들은 기능적인 척도들(바꾸어 말하면 부가처리식들에서 결수들)을 유도하고 조정설비들을 측정하는데서 가장 좋은 위치에 있기때문이다. 리상적으로 이 부가처리식들은 PVM과 MPI와 같은 그러한 표준들에 의해서 모든 연산들을 정량화해야 한다. 최소한 모든 호상작용연산들을 정량화해야 한다. 이 단계에서 일반적인 평가기준들이 아래에서 고찰된다. 또한 이 평가기준들은 부가처리식을 얻어 내는데 리용된다.

부가처리측정조건

측정실험들을 구성하는 엄밀한 조건들을 명백히 하여야 한다.

- 사용되는 자료구조

- 프로그람작성언어서고 그리고 사용되는 콤파일러선택
- 일반적으로 부가처리특징은 묶음처리방식으로 수행되여야 한다. 왜냐하면 이것은 대부분의 제품들이 실행되는 방식이기때문이다.
- 사용되는 통신하드웨어와 규약
- 시계시간을 측정하겠는가 또는 CPU시간을 측정하겠는가 하는것인데 일반적으로 시계시간이 더 쓸모 있다.

실례 3.10. 400개 마디 IBM SP2의 통신부가처리

[655]에서 256개의 마디까지에 대한 IBM SP2 MPP의 통신부가처리를 측정하였다. 측정조건들은 다음과 같다.

- 리용된 자료구조는 언제나 마디기억에 충분히 적합하도록 작게 만들어 비용이 드는 페지고장이 일어 나지 않도록 한다.
- 검사프로그람은 표준 C로 작성한다. 리용된 서고함수는 표준 I/O, 시간함수 그리고 MPI(혹은 MPL)통신함수들이다. 다른 서고함수 혹은 아쎔블러코드는 허용하지 않는다.
- 가장 좋은 콤파일러선택이 언제나 리용된다. 즉 mpcc−O_3−9arch=pwr2_C프로그람콤파일러 mpcc는 사실상 IBM콤파일러를 호출하고 MPI와 MPL서고를 련결하는 script파일이다.
- 체계자원을 가능한껏 리용하여 조작체계와 다른 리용자처리에서의 간섭을 최소화한다. Host,list파일 즉 유일하게 주어 진 host이름에 기록함으로써 매 256개 마디들은 한프로쎄스에 의해 혼자서 쓰일수 있는 모임이다.
- SP2은 Ethernet 혹은 소유권 고성능절환기(HPS)를 통하여 통신을 진행한다. 매 통신매체에 대하여서는 표준인터네트규약(IP규약) 혹은 IBM전개된 사용자공간규약(US규약)을 통하여 통신을 실현할수 있다. 가장 좋은 통신성능을 얻기 위하여 US규약을 통한 HPS를 리용하는 다음과 같은 환경변수들을 선택할수 있다.

```
Setenv  EUIDEVICE  csso/*(Ethernet 대신에 HPS리용)*/
Setnv  EUILIB  us/*(Ip대신에 OS규약리용)*/
```

시계시간과 CPU시간(체계시간과 사용자시간)의 두가지는 검사결과에서 믿음성을 얻기 위해 측정한다. Unix시간함수는 CPU시간을 측정하기 위해 리용되고 gettimeofday()함수는 시계시간을 측정하기 위해 리용된다.

부가처리측정방법 부가처리를 측정하는것은 간단해 보이지만 정확한 측정결과를 얻는것은 매우 힘든 일이다. 이것은 세가지 리유에 기인한다.

첫째로, 대부분의 콤퓨터체계들은 조잡한 시간측정기분해를 마이크로초 지어는 미리초순서로 보장한다.

둘째로, 병렬콤퓨터 특히 MPPs와 클라스터들에서 처리기들은 흔히 비동기적으로 동

작하므로 일반박자에 따르지 않는다. 이것은 시간동기화문제[1]를 일으킨다. 즉 처리기들이 동시에 동작하도록 하는것은 대단히 힘들다.

셋째로, 측정결과들은 지어는 같은 통신동작에 대하여서도 서로 다르게 변할수 있다. 점대점통신(실례로 마디 0과 마디 1사이)에 대한 일반화된 방법은 Ping−Pong도식이다.

마디 0은 매개 byte의 통보문을 마디 1로 보내는 보내기동작을 실행하며 마디 1은 통보문을 얻기 위하여 대응하는 받기를 실행한다. 그다음 마디 1은 즉시 같은 통보문을 마디 0으로 되돌려 보낸다.

이 Ping−Pong동작의 전체 시간을 점대점통신시간을 얻기 위하여 2로 나눈다. 즉 단 하나의 보내기 혹은 받기동작을 실행하는 시간이 바로 점대점통신시간이다.

이 절차를 다음과 같은 실례로써 설명한다.

실례 3.11. 지연시간을 측정하는 Ping−Pong도식

Ping−Pong도식은 아래와 같이 서술한다.

```
for(i=0;i<Runs;i++)
if(my_node_id==0){                                  /*보내기*/
Tmp=Second();
Start_time=Second();
Send an m−byte message to node 1;
Receive an m−byte message from node 1;
End_time=second();
Timer_overhead=start_time-tmp;
Total_time=end_time−start_time−timer_overhead;
Communication_time[i]=total_time/2;
 } else if (my_node_id ==1) {                       /*받기*/
receive an m-byte message from node 0;
send an m-byte message to node 0;
}
}
```

Runs고리는 통신시간을 얻는데 쓰이며 그로부터 최대, 최소, 평균값들을 계산할수 있다. Ping−Pong도식은 시간동기화문제에 의해 영향을 받지 않는다. 때문에 시간측정기함수 Second()는 다만 한개 처리기(마디 0)에서 실행된다.

Ping−Pong도식의 일반화는 hot_potato(fire_brigade로 알려 진) 방법이다.

두개 마디(보내기와 받기)대신에 *n*개 마디를 포함한다.

마디 0은 *m*byte통보문을 마디 1에 보내고 마디 1은 즉시 같은 통보문을 마디 2로 보낸다. 결국 마디 *n*−1은 통보문을 마디 0으로 되돌려 보낸다.

실례 3.12. 집중통신성능측정

분산기억기다중콤퓨터의 *n*개의 매 마디들에 의해 실행되는 다음과 같은 SPMD프로그람을 고찰하자.

장벽은 측정처리에서 비동기동작을 동기화하는데 적용한다.

```
for(i=0;i<Runs;i++){
    barrier  synchronization;
    tmp=Second();
    start_time=Second();
    for(j=0;j<Iterations;j++)
            the_collective_routine_being_message;
    end_time=Secnod();
    timer_overhead=start_time_tmp
    total_time=end_time−start_time_overhead
    local_time=total_time/Iterations
    communication_time[i]=maximum  of  all  n  local_time values;
}
for ( i=0 ; i < Runs ; i++) {
    if (my_node_id ==0) {
        tmp = Second();
        start_time = Second();
    }
    node 0 broadcasts an empty message to all n nodes ;
    for ( j=0 ; j< Iterations ; j++)
        the_collective_routine_being_measured ;
    all nodes perform an empty reduction to node 0 ;
    if (my_node_id ==0) {
        end_time = Second();
        timer_overhead = start_time - tmp ;
        communication_time[i] =end_time - start_time - timer_overhead;
    }
}
```

장벽동기화는 마디를 시간동기화하려고 시도하는데 실현될수 없다. 장벽은 국부동기화동작이지 시간동기화동작은 아니다.

결론적으로 마디들은 tmp=Second()명령이 동시에 실행하도록 반드시 출발하지 않는다. 도식은 어느것이 집중동작을 위한 Ping−Pong방법을 일반화하는가 하는것을 보여 준다.

이 방법은 마디 0에서만 시간을 계수함으로써 시간동기화문제를 피한다. 방송동작은 모든 마디들을 출발하게 하며 감소동작은 모든 마디를 끝나게 한다.

방송동작전에 마디는 반복고리실행을 시작할수 없다. 감소동작후에 모든 마디들은 반복고리를 끝낸다.

반복값이 충분히 크면 방송동작과 감소동작에 대해 일어 난 장애(교란)는 무시할수 있다.

부가처리표현 측정으로부터 부가처리자료를 얻은 후 그것들을 어떻게 해석하는가? 다음과 같은 세가지 해석방법들이 제기된다.

표 3-17 SP2에서 한통로 점대점통신시간

통보문크기	기 계 크 기 n			
m	2	8	32	128
48	46	47	48	
1KB	101	120	120	133
64KB	1969	1948	1978	2215
4MB	1.2M	1.2M	1.2M	1.2M

방법 1 표에 자료를 제시한다. 실례로 표 3-17은 소유 MPL통신서고를 동작시킴으로써 SP2에서 점대점통신에 대한 시간측정결과를 보여 준다. 그것은 마디 0이 m byte통보문을 마디 n-1로 보내는 (μs)부가처리를 보여 준다. 이 방법은 가장 정확하고 상세한 정보를 제시한다. 그러나 이것은 사용자에 대하여 편리하지 않다.

방법 2 자료를 그림 3-5에서 보여 주는바와 같이 곡선으로 제시하자. 우점은 곡선이 통신부가처리증가경향을 보여 줄수 있다는것이다.

방법 3 리상적으로 사용자는 여러가지 통보문길이와 기계크기에 대한 부가처리를 평가하는데 리용할수 있는 단순닫긴형표현을 쓰는것을 더 좋아 하지만 바로 측정된것들은 그러한것이 아니다.

실례로 측정된 시간자료의 최소평방정합을 리용하여 우리는 SP2에서 점대점통신부가처리를 통보문길이의 선형함수로써 표현할수 있다. 즉 t=46+0.035$m\mu$s.

실제로 진행하면 곡선정합에 의해 일어 난 불정확성은 그림 3-5에서 보여 주는바와 같이 작다.

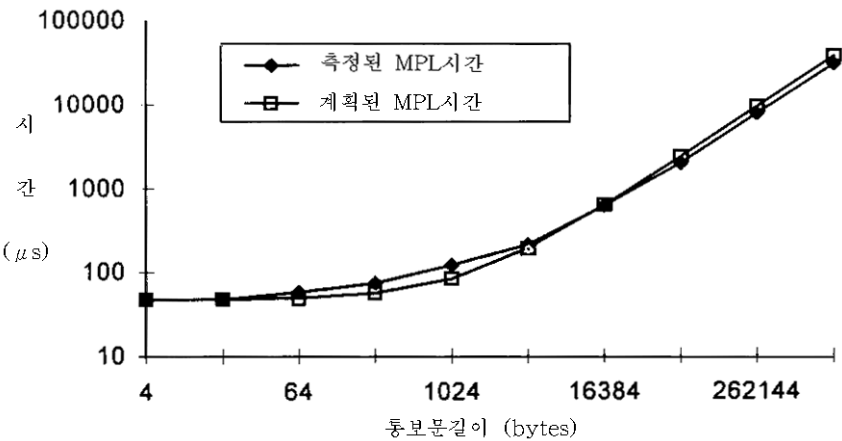



그림 3-5. SP2에서 점대점통신의 부가처리

병렬성부가처리 많은 병렬처리응용에서 동일한 병렬프로그람이 생자료를 처리하기

위해 여러번 실행된다. 모든 프로쎄스들과 그룹들을 창조하는 한번의 부가처리가 있는데 그것은 다음 계산에 리용할수 있다. 다른 말로 병렬성부가처리는 해제된다.

프로쎄스나 그룹을 창조하는것은 병렬콤퓨터 특히 MPPs와 클라스터에서 비용이 든다. 측정결과[655]를 보면 SP2에서 프로쎄스를 창조하는것은 약 10000μs 혹은 그이상이 걸리며 수백만개의 류동소수점연산과 등가이다.

그룹창조는 1ms걸린다.

많은 프로쎄스그룹을 창조하는데 필요한 프로그람은 큰 grain크기를 가져야 한다.

점대점통신 Hockey[315]는 점대점동작에 대한 통신시간(μs)을 특징 짓는 모형을 아래와 같이 제기하였다. 여기서 통신부가처리 $t(m)$은 통보문길이 m(byte에서)의 선형함수이다. 즉

$$t(m) = t_0 + m / r_\infty \tag{3.3}$$

t_0은 μs단위의 시동시간, r_∞는 MB/s단위의 접근대역너비이다.

두개의 추가적인 파라메터들은 Hockvey에 의해 도입되였다. $m_{1/2}$ byte로 표시되는 피크절반길이는 점근대역너비의 절반을 얻는데 요구되는 통보문길이이다.

π_0 MB/s로 표시되는 특수한 성능은 짧은 통보문에 대한 대역너비를 가리킨다. 4개의 파라메터 $r_\infty, t_0, m_{1/2}, \pi_0$에서 두개만 독립이고 다른 두개는 다음과 같은 관계 $t_0 = m_{1/2} / r_\infty = 1/\pi_0$을 써서 유도할수 있다. 파라메터 $m_{1/2}$은 체계가 짧은 통보문통신을 얼마나 잘 지원하는가를 보여 주는 량이다.

SP2에 대하여 $t(m) = 46 + 0.035m$이다. 다른 말로 시동부가처리는 $t_0 = 46\mu$s이고 점근대역너비는 $r_\infty = 1/0.035 = 28.57$MB/s이며 최대절반의 통보문길이는 $m_{1/2} = t_0 \times r_\infty = 1314$이다.

집중통신 우리는 다음과 같은 집중통신을 측정하였다.

방송동작에서 처리기 044은 $m$byte통보문을 모든 n처리기들에 보낸다. 수집동작에서 처리기 0은 n개의 매 처리기들로부터 $m$byte통보문을 받으므로 마지막에는 mnbyte를 받는다.

산란동작에서 처리기 0은 명백한 $m$byte통보문을 n개의 매 처리기들에 보내며 마지막에는 처리기 0에 의해 mnbyte가 보내여 진다.

전체 교환동작에서 매 처리기는 명백한 $m$byte통보문을 n개의 매 처리기들에 보내여 마지막에는 mn^2byte들이 통신된다.

순환밀기동작에서 처리기 i는 $m$byte통보문을 처리기 i+1에 보내고 처리기 n−1은 $m$byte를 처리기 0으로 되돌려 보낸다.

현재 통보문넘기기체계에서 집중통신은 통보문을 자기자체에 보내는 프로쎄스를 언제나 요구한다는것을 주목하자.

그것은 mn^2byte가 전체 교환에서 통신되지만 $mn(n-1)$byte는 통신되지 않기때문이다.

우리는 식 (3.3)의 Hockey의 표현을 아래와 같이 확장하였다. 즉 통신부가처리

$T(m,n)$은 m과 n개의 함수이다. 그러나 시동지연시간은 n에만 의존한다.

점근대역너비 $r_\infty(n)$은 역시 기계크기 n에 따라 변하며 식 (3.3)에서와 같이 더는 상수가 아니다.

$$T(m, n) = t_0(n) + m / r_\infty(n) \qquad (3.4)$$

표 3-18 SP2에 대한 집중통신부가처리표현

동작	시간측정식
방송	(52logn)+(0.029logn)m
수집/산란	(17logn+15)+(0.025n-0.02)m
전체 교환	80logn+(0.03$n^{1.29}$)m
순환밀기	(6logn+60)+(0.003logn+0.04)m

측정된 시간측정자료를 $t_0(n)$와 $r_\infty(n)$의 여러가지 형식에서 정합한후에 표 3-18에서 보여 주는바와 같이 5개의 집중동작에 대한 식을 유도하였다.

집중계산 우리는 3개의 대표적인 집중계산동작 즉 장벽, 감소, 주사를 측정하였다.

곡선정합부가처리표현들은 표 3-19에서 보여 준다.

표 3-19 SP2에 대한 집중계산부가처리

동작	시간식
장벽	94logn+10
감소	20logn+23
주사	60logn-25

256개 처리기이상에 대한 장벽부가처리는 768μs이며 768×266=202692flop(류동소수점연산)만큼 실행하는 시간과 등가이다.

이것은 동기알고리듬을 왜 리용해야 하는가 하는 질문에 대답한다. 그 대답은 grain 크기가 클 때뿐이다.

이 방법[656]은 정확성을 얻기 위해 매 동작을 개별적으로 량자화하는 방법을 요구한다. 그러나 부가처리표현은 주어 진 콤퓨터가동환경에 대하여 단 한번만에 측정되고 유도될것을 요구하며 사용자공동에 의해 여러번 리용될수 있다.

방법을 그림 3-6에서 짧은 통보문과 긴 통보문 두가지에 대해 예측을 진행하여 비교하였다.

그림 3-6의 ㄱ)에서는 mn^2=16MB(실례로 m=1024byte, n=128)일 때 두 방법의 상대적 오차를 보여 준다. 전통적인 방법은 특히 많은 수의 처리기들에 대하여 부가처리를 파소평가한다.

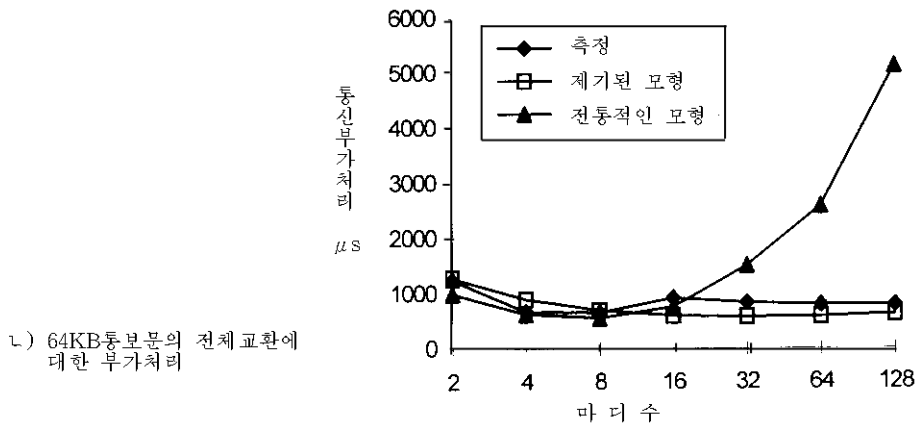
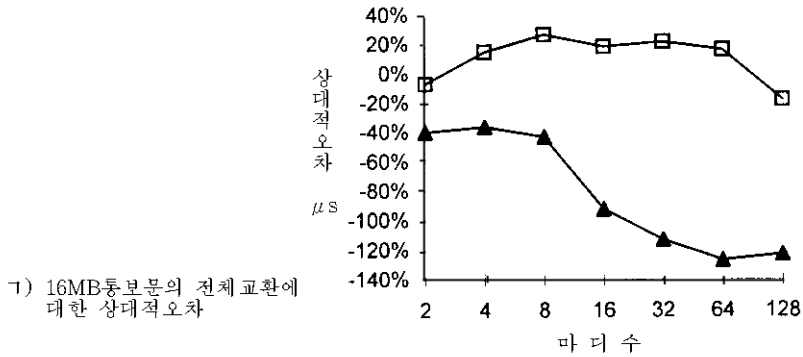



그림 3-6. 전체 교환부가처리를 예측하는 두가지 방법

ㄴ)에서는 mn^2=64KB(실례로 n=128일 때 m=4byte)일 때 측정된 부가처리를 두 방법에 의해 실행된것과 비교한다.

우리의 방법은 측정된 결과에 접근하였으며 전통적인 방법은 대형기계들에 대하여 부가처리를 명백히 과대평가하였다.

3. 5. 병렬프로그람의 성능

이 절에서 우리는 병렬리용에 대한 많은 성능론점들과 척도를 고찰한다.

우리의 목적은 해석을 례를 들어 설명하기 위해 위상병렬모형리용에 기초한다.

제기된 모든 성능척도는 다른 형태의 병렬프로그람에 적용해도 마찬가지이다.

3. 5. 1. 성능척도

k개의 주요한 계산위상수렬 $C_1, C_2, \cdots, C_k$로 이루어 진 련속프로그람 C를 고찰하자. 우리는 병렬성정도 DOP_i를 가진 매 위상 C_i를 써서 C로부터 효과적인 병렬프로그람을 전개하려고 한다. 병렬성과 호상작용은 둘 다 부가처리를 도입한다. 위상병렬프로그람을 그림 3-7에 묘사하였다.

우리는 병렬프로그람의 출발에서 덩어리병렬성부가처리가 있다고 가정한다.

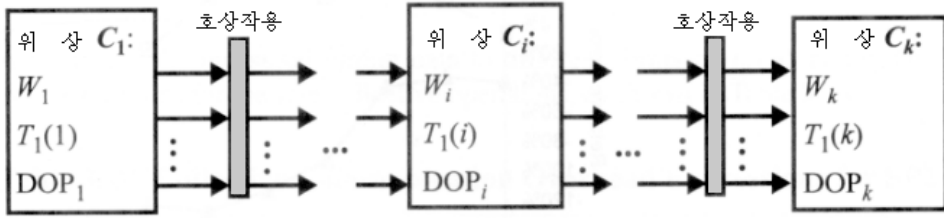



그림 3-7. 응용프로그람알고리듬의 위상병렬모형

기본척도 의미론적으로 한개 위상(걸음)의 모든 동작들은 다음걸음이 시작되기전에 끝나야 한다.

걸음 C_i는 W_i백만개 류동소수점연산(Mflop)의 계산적작업부하를 가지며 한개 처리기에서 실행시간은 $T_1(i)$이다. 그것은 DOP_i의 병렬성정도를 가진다.

다른 말로 말하면 $1 \le n \le DOP_i$인 n개 처리기에서 실행할 때 걸음 C_i에 대한 병렬실행시간은 $T_n(i) = T_1(i)/n$이다.

N개 마디에 관한 전체 병렬실행시간은 다음과 같다.

$$T_n = \sum_{1 \le i \le k} \frac{T_1(i)}{\min(DOP_i, n)} + T_{par} + T_{interact} \quad . \tag{3.5}$$

여기서 T_{par}와 $T_{interact}$는 각각 모든 병렬성과 호상작용을 표시한다.

극값척도 T_n, P_n에 대한 하계(아래제한)와 상계(웃제한)를 주는 몇개의 극값척도가 있다.

T_∞를 림계경로의 길이라고 하자. 그것은 제한되지 않는 마디수를 리용하고 모든 병렬성과 호상작용부가처리를 배제하는 응용프로그람실행시간과 등가이다.

식 (3.5)로부터

$$T_\infty = \sum_{1 \le i \le k} \frac{T_1(i)}{DOP_i} \tag{3.6}$$

$T_n = T_\infty$가 성립하는 가장 작은 n을 **최대병렬성**이라고 부르고 $N_{\max}$로 표시한다. 이

것은 실행시간을 감소하는데 리용할수 있는 최대마디수이다.

이 척도는 $N_{max} = \max_{1 \le i \le k}(DOP_i)$를 써서 계산할수 있다. 지속된 속도 P_n은 그것의 상계(웃제한)로서 최대성능 $P_\infty = W/T_\infty$를 가진다. N번째 마디실행시간 T_n은 $T_{1/n}$와 T_∞의 하계(아래제한)를 가진다. 즉

$$T_n \ge \max(T_1/n, T_\infty) \tag{3.7}$$

평균병렬성 T_1/T_∞는 속도에서 상계(웃제한)를 제공한다. 즉 $S_n \le T_1/T_\infty$

Brent는 모든 병렬성과 호상작용부가처리를 배제함으로써 T_n이 다음과 같은 부등식으로 제한된다는것을 증명하였다[109].

$$T_1/n \le T_n < T_1/n + T_\infty \tag{3.8}$$

식 (3.7)과 통합하면

$$\max(T_1/n, T_\infty) \le T_n < T_1/n + T_\infty \tag{3.9}$$

표 3-20 위상병렬모형에 기초한 성능척도

표기	용어	정의
T_1	순차시간	$T_1 = \sum_{1 \le i \le k} T_1(i)$
T_n	병렬시간, n-마디시간	$T_n = \sum_{1 \le i \le k} \frac{T_1(i)}{\min(DOP_i, n)} + T_{par} + T_{interact}$
T_∞	림계경로	$T_\infty = \sum_{1 \le i \le k} \frac{T_1(i)}{DOP_i}$
P_n	n-마디속도	$P_n = W/T_n$
S_n	n-마디속도증가	$S_n = T_1/T_n$
E_n	n-마디효률성	$E_n = S_n/n = T_1/(nT_n)$
U_n	n-리용	$U_n = P_n/(nP_{peak})$
T_o	전체부가처리	$T_o = T_{par} + T_{interact}$
	평균병렬성	T_1/T_∞
	평균부가처리	T_o/W
	평균립도	W/T_o

이 부등식들은 병렬실행시간을 평가하는데 쓸모 있다.

다른 척도 평균립도는 W/T_0으로 정의한다.

여기서 W는 전체 작업부하이며 $T_0 = T_{interact} + P_{par}$는 모든 병렬성과 호상작용부가처리를 포함하는 전체 부가처리이다.

역(reciprocal)척도를 **평균부가처리**라고 부르며 T_0/W로서 정의한다. 많은 경우에 전체 부가처리는 통신부가처리에 의해 결정된다.

이 척도를 표 3−20에 종합한다.

실례 3.13. 병렬 APT성능평가기준프로그람의 성능척도

APT프로그람에 대한 조잡한 (coarse−grain)위상병렬알고리듬을 그림 3−8에서 보여준다. 파라메터는 N=256이다.

DP단계는 8192개 마디까지 올라 가 분포될수 있다. 전체 교환단계는 통보문의 총 길이가 17MB인 모든 마디에 의해 실행될수 있다.

HT단계는 단일마디 (DOP=1)에서 련속적으로 실행된다.

방송동작은 80KB통보문을 모든 마디에 보낸다.

BF단계는 256개 마디를 리용하는 beamforming을 수행한다. TD단계는 256개마디 에서 국부적으로 목표를 추적하는데 필요하다.

그러면 모든 국부목표기록들은 최종집중감소동작을 통하여 합쳐 진다.

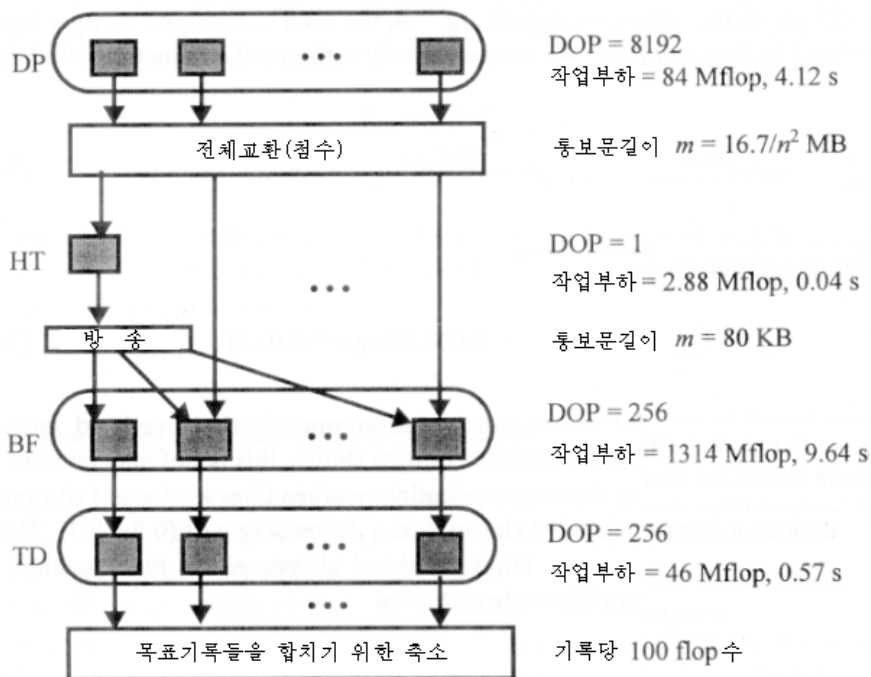



그림 3−8. STAP에서 병렬 APT알고리듬구조

매 계산단계의 작업부하를 표 3−12에 제시하는데 한SP2마디에서 Mflop와 실행시간을 보여 준다.

병렬부가처리는 무시할수 있다. 왜냐하면 정적과제작성이 SP2에서 리용되였기때문이다.

성능척도의 극값을 예측하여 모든 통신부가처리를 무시할수 있다. 이것을 **령부가처리예측**이라고 부른다. 그림 3−8로부터 최대병렬성은 max(8192, 1, 256, 256)=8192, 전체 작업부하는 $W=1447$ Mflop이고 련속실행시간은 $T_1=14.37$ s, 림계경로는 다음과 같다.

$$T_\infty=\frac{4.12}{8192}+\frac{0.04}{1}+\frac{9.64}{256}+\frac{0.57}{256}=0.08\text{s}$$

따라서 최대성능은 $P_\infty=W/T_\infty=1447/0.08=18087$ Mflop/s 이고

평균병렬성은 $T_1/T_\infty=14.37/0.08=180$ 이다.

실례 3.14. APT성능평가기준에서 호상작용부가처리평가

우리는 SP2에서 병렬APT프로그람이 동작할 때의 호상작용부가처리 $T_{interact}$를 평가하기 위하여 표 3−18과 표 3−19에서의 부가처리표현을 리용할수 있다.

호상작용부가처리는 3개 통신의 통합이다. 즉

$$T_{interact}=T_{index}+T_{ncast}+T_{reduce}$$

표 3−18과 그림 3−8로부터 16.7MB의 전체 교환부가처리는 다음과 같다.

$$T_{index}=80\log n+0.03n^{1.29}m\mu\text{s}=0.00008\log n+0.5n^{-0.71}\text{seconds} \qquad (3.10)$$

Broadcast 부가처리는

$T_{bcast}=52\log n+(0.029\log n)m\mu\text{s}=0.00237\log n$ seconds 로 표현할수 있다.

표 3−19로부터 nflop(류동소수점연산)수를 감소하는 시간은 n개의 매 마디로부터 $20\log n+23\mu\text{s}$ 이다.

그림 3−8의 감소단계에서 n개 목표기록들을 결합하는것이 필요하며 그러면 매개는 100flop수를 가진다.

감소부가처리를 보수적으로 평가하면 다음과 같다.

$$T_{reduce}=100(20\log n+23)\mu\text{s}=0.002\log n+0.0023\ \text{seconds} \qquad (3.11)$$

그러면 전체 호상작용부가처리는

$$T_0=T_{interact}=0.5n^{-0.71}+0.00445\log n+0.0023\,. \qquad (3.12)$$

병렬처리에서 일반적인 견해는 더 많은 마디들이 쓰일 때 통신부가처리가 증가한다는것이다.

우의 실례가 보여 주는바와 같이 이러한 견해는 거짓이 될수 있다.

APT프로그람에서 통신부가처리는 상수요소(0.0023), 증가부분(0.0045logn), 감소부분($0.5n^{-0.71}$)을 가지고 있다.

망효과는 256개 마디이상 쓰이지 않을 때 전체 통신부가처리의 기계크기가 증가하는데 따라 감소한다는것이다.

실례 3.15. APT성능평가기준의 기대실행시간

우리는 n마디 SP2에서 그림 3−8의 병렬 APT알고리듬의 실행시간 T_n을 예측한다. 여기서 $n \le 256$ 이다.

또한 256개 마디가 쓰일 때 평균립도를 계산한다. 자세한 유도는 련습(문제 3.6)으로 남긴다.

n마디를 리용하는 전체 실행시간은 다음과 같다.

$$T_n = T_{comp} + T_{par} + T_{interact}$$
$$= \frac{14.33}{n} + 0.5n^{-0.71} + 0.00445\log n + 0.0423$$

전체 작업부하는 W=1447Mflop 혹은 SP2마디에서 14.37이다. 평균립도는

$$\frac{W}{T_0} = \frac{1447}{0.0479} = 30209\,. \qquad . \qquad (3.13)$$

평균적으로 256개 마디들은 통신의 매 us에 대하여 30,209flop(류동소수점연산) 혹은 통신의 매 μs에 대하여 마디당 30209/256=118flop이다.

평균부가처리는 통신의 매 Mflop에 대해 1/30209=33us이다.

또한 작업부하로써 실행시간을 리용할수 있다. 그러면 평균립도는

$$\frac{W}{T_0} = \frac{14.37}{0.0479} = 300\,.$$

그리하여 평균적으로 256개 마디들은 통신의 매초에 대하여 300s의 계산을 혹은 통신의 매초에 대하여 마디당 300/256=1.17s의 계산을 진행한다.

3. 5. 2. 성능평가기준에서 유효병렬성

응용프로그람에는 잠재적인 병렬성의 넓은 범위가 있다(병렬성할 가능성이 아주 풍부하다.).

공학과 과학코드(code)들은 자료병렬성으로 인하여 높은 DOP를 공개한다. Kumar(1988)는 계산성이 강한 코드들은 리상화된 환경에서 매 박자주기에서 동시에 500~3500산수연산을 실행할수 있다는것을 제출하였다.

그러나 명령준위의 병렬성은 훨씬 낮다.

Wall[632]는 명령준위의 병렬성한계가 5근방이며 드물게는 7을 초과한다는것을 지적하였다.

Bulter 등(1991)은 모든 제약들이 제거되였을 때 어떤 과학적프로그람들에서 IPL은 주기당 17개 명령을 초과할수 있다는것을 보고하였다.

표 3-21 PERFECT성능평가기준프로그람의 평균병렬성

프로그람	병렬성	프로그람	병렬성	프로그람	병렬성
ADM	10.8	FLO52	206.9	QCD	2.4
ARC2D	336.0	MDG	5.3	SPEC77	13.8
BDNA	139.5	MG3D	1.3	TRACK	38.7
DYFESM	17.9	OCEAN	272.4	TRFD	87.9

일부 추적결과들은 구성방식과 콤파일러를 잘 구성하면 합리적으로 설계된 초스칼라처리기에서 주기당 2.0~5.8명령의 ILP를 예상할수 있다.

12개의 프로그람에 대한 PERFECT성능평가기준을 표 3-21에 준다.

실례 3.16. 기호수자프로그람에서 순환고리병렬성

Larus[392]는 표 3-22에서 보여 주는바와 같이 6개의 실지 프로그람에서 평균병렬성을 추적하였다.

고리의 병렬실행만을 고찰하면 무한히 많은 처리기들이 유효하다고 가정하고 모든 병렬성과 호상작용부가처리를 무시한다.

Gcc, xlosp, espresso들은 기호리용이고 sgefw와 dcgc는 수자리용이며 costScale은 두개 클라스의 결합이다.

작업부하렬은 프로그람을 순차적으로 실행하는데 필요한 백만박자주기수를 보여준다.

모든 란들은 모든 고리-연장의존성들이 동기화되는데 필요할 때 평균병렬성을 보여 준다. 명령은 그것의 연산수들이 모두 계산될 때까지 대기하여야 한다. 표준란은 평균병렬성을 보여 주며 콤파일러최량화에 의해 쉽게 제거될수 있는 그것의 의존성들은 무시한다.

없음 란은 모든 의존성들을 무시하는바 그것은 리상화된 경우이다.

표 3-22 6개 프로그람에서 순환고리병렬성

프로그람	기억기요구(MB)		작업부하 (Mflop)
	단순정확도	배정확도	
APT(min)	0.23	0.46	5
API(norm)	18.35	36.71	1446
APT(max)	3276.80	6553.60	12,100,000
HO(min)	0.15	0.30	21
HO(norm)	50.33	100.66	12,852
HO(max)	3276.80	6553.60	33,263,288
GEN(min)	0.26	0.52	6
GEN(norm)	100	200	5326
GEN(max)	33,957	67,914	4,604,011

실례 3.17. 3개 STAP성능평가기준프로그람의 성능

표 3-23은 STAP성능평가기준에서 3개 프로그람의 어떤 성능척도를 보여 주는데 최소, 최대 그리고 정규자료모임들을 리용한다.

입력자료크기와 작업부하는 STAP성능평가기준명세서에 의해 주어 진다[101].

최대병렬성은 개별적단계에서 최대인 DOPs를 찾아 간단히 계산한다. 림계경로는 가능한 무한히 많은 마디들이 리용될 때의 실행시간이다. 간단히 고찰하기 위해 매 류동소수점연산은 동일한 시간이 걸린다고 가정하자.

그러면 매 단계의 림계경로는 그것의 작업부하를 DOP로 나눈것이다.

평균병렬성은 림계길에 대한 작업부하의 비률이다.

실례로 표 3-12로부터 HO-PD프로그람은 3개 단계 즉 Doppler처리, Beamforming 그리고 목표검출을 가진다. 명사자료모임에 대해 이 단계들은 각각 49152, 256 그리고 256의 DOP를 가진다.

최대병렬성은 max(49152, 256, 256)=49152로 계산된다.

표 3-23 3개의 STAP성능평가기준프로그람의 성능

프로그람	입구자료크기 (MB)	작업부하 (Mflop)	최대병렬성	평균병렬성	림계경로 (Mflop)
APT (min)	0.23	5	1800	10	0.51
APT (norm)	18.35	1446	8192	177	8.19
APT (max)	3276.80	12,100,000	400,000	1005	12,036.05
HO (min)	0.15	21	1176	17	1.24
HO (norm)	50.33	12,852	49,152	261	49.35
HO (max)	3276.80	33,263,288	200,000	65,839	505.22
GEN (min)	0.26	6	2048	103	0.05
GEN (norm)	100	5326	196,608	108	49.27
GEN (max)	33,957	4,604,011	16,580,608	4332	1062.81

병렬 HO-PD프로그람의 림계경로는 값 220/49152 +(12618+14)/ 256= 49.35Mflop로 계산된다.

평균병렬성은 비률 12852/49.35=261로 계산된다.

우에서 보여 준 유효병렬성의 측정들은 지어 기본블로크경계들이 무시될 때에도 비수값계산이 상대적으로 작은 병렬성을 가진다는것을 보여 준다.

기본블로크는 단일입력점과 단일출구점을 가진 렬 혹은 명령블로크이다. 콤파일러최량화와 알고리듬재설계는 응용프로그람에서 유효병렬성을 증가시킬수 있다. 병렬성추출을 기본블로크로 제한하는것은 보통 프로그람에서 가능한 ILP를 약 2~5의 인수로 제한하기 위해서이다.

그러나 DOP는 다중처리기들이 기본블로크들의 경계를 넘어 서 병렬성을 개발하기 위해 동시에 쓰일 때 수천개의 과학코드로 넓힐수 있다.

3. 6. 확대가능성과 속도증가해석

이 절에서 우리는 병렬콤퓨터프로그람결합의 확대가능성을 해석하고 예측하기 위하여 앞절들에서 전개한 척도를 어떻게 리용하는가를 고찰한다.

우리는 세가지 속도증가척도에 기초한 세가지 성능모형을 도입한다. 그다음 확대가능성해석에 대한 3가지 방법을 고찰하는데 그것들은 각각 상수효률, 상수속도 그리고 상수리용에 기초한다.

3. 6. 1. Amdahl의 법칙: 고정된 문제크기

실시간응답을 요구하는 많은 실천적응용프로그람들에서 계산적작업부하 W는 자주 고정된다.

처리기수가 병렬콤퓨터에서 증가하는데 따라 고정된 작업부하를 병렬실행하기 위해 더 많은 처리기들에 분배한다.

그러므로 주요목적은 가능한껏 빨리 결과를 생성하는것이다.

고정된 작업부하 W를 가진 문제를 고찰하자.

작업부하 W를 두개 부분 즉 $W=\alpha W+(1-\alpha)W$ 로 나눌수 있다고 가정하자. 여기서 W의 α퍼센트는 순차적으로 실행되여야 하며 나머지 $1-\alpha$퍼센트는 n개 마디들에 의해 동시에 실행될수 있다.

모든 부가처리들이 무시된다고 가정하면 고정부하속도증가는 $n\to\infty$일 때

$$S_n=\frac{W}{\alpha W+(1-\alpha)(W/n)}=\frac{n}{1+(n-1)\alpha}\to\frac{1}{\alpha}\ \text{as}\,n\to\infty \qquad (3.14)$$

로 정의한다.

이 식을 Amdahl의 법칙이라고 부르는데 병렬체계를 연구하는 가장 기본적인 법칙의 하나이다.

Amdahl법칙은 다음과 같은 몇가지 의미를 가지고 있다.

(1) 주어 진 작업부하에 대하여 최대속도증가는 $1/\alpha$의 웃한계를 가진다. 다른 말로 말하면 프로그람의 순차구성요소는 병목문제이다. 이 α는 프로그람의 순차병목문제로서 알려 져 있다. α가 증가할 때 속도증가는 비례적으로 감소한다.

(2) 좋은 속도증가를 얻기 위하여서는 순차병목문제 α를 가능한 작게 취하는것이 중요하다.

(3) 문제가 우에서 언급한 두가지 부분으로 이루어 질 때 보다 큰 부분이 더 빨리 실행되도록 하여야 한다.

이제 앞에서 말한 모든 부가처리들을 Amdahl법칙에 넣자.

식 (3.18)에서 고정-부하속도증가 S_n은 $n \to \infty$일 때

$$S_n = \frac{W}{\alpha W + (1-\alpha)(W/n) + T_0}$$
$$= \frac{n}{1+(n-1)\alpha + \frac{nT_0}{W}} \to \frac{1}{\alpha + \frac{T_0}{W}} \text{ as } n \to \infty \qquad (3.15)$$

로 된다.

이 확장된 Amdahl법칙은 순차병목문제를 감소시킬뿐아니라 부가처리의 상반되는 영향을 줄이기 위하여 평균립도를 증가시켜야 한다는것을 말하여 준다. 다른 말로 병렬프로그람의 성능은 순차병목문제에 의해 제한될뿐아니라 평균부가처리에 의해서도 제한된다.

실례 3.18. 병렬APT실행의 속도증가에 대한 웃한계

그림 3-8에 있는 병렬 APT프로그람을 고찰하자.

모든 부가처리를 무시하는 (1)과 모든 부가처리들이 $T_0(\infty) \to T_0(256)$이라고 가정하는 (2)에 의해 속도증가에 관한 Amdahl 웃한계를 평가하자.

(1) 모든 부가처리들이 무시될 때 작업부하는 W=14.375와 같다. 련속구성요소는 HT단계이며 그것은 $\alpha = 0.04/13.7 = 0.278\%$이다. Amdahl법칙에 의해 속도증가는 $1/\alpha = 359$로 웃한계를 가진다. 표 3-23에서 178의 평균병렬성이 Amdahl의 법칙보다 더 엄격한 웃한계를 준다는것을 주목한다. 이 속도증가에 대한 엄격한 제한은 마디의 수를 증가시켜서는 극복할수 없다.

(2) 식 (3.12)로부터 $T_0(256) = 0.0479$ s 이다. 평균부가처리는 $T_0(\infty)/W = 0.0479/14.37 = 0.00333$ 이다. 식 (3.15)에 의해 속도증가는 엄격한 웃한계값 1/(0.00278+0.00333)=163를 가진다.

고정부하속도증가는 순차병목문제 α와 평균부가처리가 증가하는데 따라 떨어 진다.

순차병목문제는 바로 체계에서 처리기수를 증가시켜서는 해결할수 없다. 이 성질은 과거 20년이상 병렬처리에 대한 비관적인 관점으로 되였다.

Amdahl법칙에서 주요한 가정은 문제크기(작업부하)가 고정되고 기계크기가 증가하는데 따라 문제크기를 유효한 계산능력에 맞게(비례적으로) 확대할수 없다는것이다. 이것은 종종 큰 규모의 체계가 작은 문제를 푸는데 리용될 때 리득감소로 이끌어 간다.

다음절에서 이 문제에 대한 해결을 고찰한다.

3. 6. 2. Gustafson의 법칙: 고정된 시간

순차병목문제는 고정문제크기제한을 제거함으로써 완화시킬수 있다. John Gustafson(1988)은 기계크기가 증가하는데 따라 문제크기를 확대함으로써 개선된 속도증가를 얻는 고정시간개념을 제기하였다.

더 높은 정확성을 위한 확대화 최소일감처리시간이상의 정확성을 강조하는 많은 응용프로그람들이 있다.

큰 계산능력을 얻기 위해 기계를 크게 하는데 따라 보다 정확한 풀이를 생성하고 적은 실행시간을 변화시키지 않으면서 보다 큰 작업부하를 얻기 위하여 문제크기를 증가시키려고 할수 있다.

대표적인 실례들로는 구조해석을 진행하는 유한요소법 리용 혹은 날씨예보에서 계산론적액체동력학문제를 푸는 유한미분법 리용을 들수 있다. 품질이 낮은 격자는 거의 계산을 요구하지 않지만 품질이 높은 격자는 보다 많은 계산을 요구하며 보다 큰 정확성을 요구하게 된다.

날씨예보모의는 종종 4차원 PDEs의 풀이를 요구한다.

매 물리차원 (X, Y, Z)에서 10개의 인수로 격자공간을 축소하고 같은 크기로 시간단계를 증가시키면 격자점은 1000배이상 증가한다.

따라서 작업부하는 적어도 1000배로 더 크게 증가한다.

그러한 문제를 확대하면 물론 같은 실행시간에 더 큰 계산력을 요구한다.

기본동기는 배수를 절약하는데 있는것이 아니라 보다 정확한 날씨예보를 얻는데 있다. 정확성을 위한 이 문제확대화는 Gustafson에게 고정시간속도증가모형을 전개시킬 동기를 주었다.

이 확대된 문제는 모든 증가된 자원들이 작업하도록 하며 보다 좋은 체계리용을 가져다 준다.

고정시간속도증가 정확성-림계응용에서 우리는 보다 작은 기계에서 작은 문제를 푸는데 걸리는 실행시간을 가지고 대형기계에서 가능한 가장 큰 문제를 풀려고 한다.

기계크기가 증가할 때 작업부하 역시 증가한다.

원래문제가 작업부하 W를 가진다고 하자.

그 가운데서 α퍼센트는 순차적이며 $1-\alpha$퍼센트는 병렬로 실행될수 있다.

단일마디기계에서 실행시간은 W이다.

n마디기계에서 작업부하를 $W'=\alpha W+(1-\alpha)nW$ 로 확대한다. n마디들에서 병렬실행시간은 여전히 W이다. 그러나 확대된 작업부하를 실행하는 순차시간은 $W'=\alpha W+(1-\alpha)nW$ 이다.

확대된 작업부하를 가진 고정시간속도증가는 다음과 같다.

$$S_n' = \frac{\text{확대된 작업부하에 대한 순차시간}}{\text{확대된 작업부하에 대한 병렬시간}} = \frac{\alpha W+(1-\alpha)nW}{W} = \alpha+(1-\alpha)n \tag{3.16}$$

이 식은 Gustafson법칙으로 알려 져 있으며 작업부하가 고정실행시간을 유지하기 위해 확대되면 고정시간속도증가는 n의 선형함수이다.

문제를 유효한 계산능력에 맞게 확대할수 있을 때 순차부분도 더는 병목문제가 아니다.

Gustafson법칙이 성립하기 위해서는 $(1-\alpha)W$에서 $(1-\alpha)nW$ 로 작업부하의 병렬가능한 부분만을 확대하는것이 중요하다.

순차부분 αW 는 그대로 둔다.

확대된 속도증가 이제 Gustafson법칙에 모든 부가처리를 넣으면 확대된 속도증가는

$$S_n' = \frac{\alpha W+(1-\alpha)nW}{W+T_0} = \frac{\alpha+(1-\alpha)n}{1+T_0/W} \tag{3.17}$$

확대된 작업부하 $W'=\alpha W+(1-\alpha)nW$ 를 가진 병렬프로그람은 원래 작업부하 W에 대한 순차시간과 같은 고정계산시간을 가진다는것에 주목한다.

병렬프로그람에 대한 전체 실행시간은 또한 부가처리 T_0을 포함한다. 이 부가처리 T_0은 n의 함수이다.

실례 3.14로부터 T_0은 증가, 감소하는 혹은 n에 관하여 상수인 성분을 가지고 있다는것을 알수 있다.

일반화된 Gustafson법칙은 부가처리를 n에 관하여 감소하는 함수로 조종함으로써 선형인 속도증가를 얻을수 있다. 그러나 이것은 흔히 어렵다.

실례 3.19. APT에서 부가처리고찰이 없는 확대된 속도증가

그림 3-8에 있는 병렬 APT프로그람을 고찰하자. 문제파라메터는 N=256이다.

작업부하는 $W=0.011N^2+2.8N+2.88$ Mflop 혹은 임의의 N에 대하여 $W=0.00016N^2+0.015N+0.04$s 로 근사화될수 있다.

N=256에 대하여 $W=147$ Mflop 혹은 14.37s이다.

128개 마디들이 리용되고 모든 부가처리들이 무시된다고 가정할 때 고정시간속도증

가를 평가하자.

또한 확대된 작업부하에서 실행되는데 얼마만한 Mflop가 필요한가를 지적하자.

모든 부가처리가 무시될 때 순차실행시간은 W=14.37s이다. 본질적으로 순차 HT단계는 전체 작업부하의 α=0.04/14.37=0.27%로 본다.

식 (3.17)로부터 확대된 속도증가는 다음과 같다.

$$S_n^{'} = \alpha + (1-\alpha)n = 0.00278 + 0.99722n = 127.65$$

확대된 작업부하에 대한 순차시간은 $\alpha W + (1-\alpha)nW = 127.65 \times 14.37 = 1834$ s이다.

달리 말하여 시간작업부하는 1834/14.37=127.65배로 확대된다.

이 값을 N에 대하여 푸는 시간작업부하표현에 대입할수 있다.

$$0.00016N^2 + 0.015N + 0.04 = 1834$$

그러면 N=3339이다.

따라서 N은 3339/256배로 확대된다.

이제 이 값을 flop작업부하표현에 대입하면 다음과 같다.

$$W = 0.011N^2 + 2.8N + 2.88 = 131996$$

즉 131996Mflops가 확대된 작업부하에서 수행되는데 필요하다. Flop작업부하는 131996/1447=91배로 확대되였다.

문제크기에 대한 주의 문제크기의 정의를 명백하게 하여야 한다.

우의 실례는 3개의 서로 다른 정의 즉 문제파라메터 N, 작업부하 혹은 시간(초)을 보여 준다.

이 정의들은 문제를 확대하는데서 중요한 차이로 된다. 시간작업부하가 127배 확대되면 flop작업부하는 91배 확대된다. 문제 파라메터 N는 13배 확대된다.

실례 3.20. 병렬 APT에서 모든 부가처리를 포함하는 확대된 속도증가

실례 3.19를 반복하되 모든 부가처리들을 고찰할 때 128개마디들에서 확대된 속도증가를 평가하자.

전체 교환단계에서 통보문길이는 $m = 256N^2/n$ byte라고 가정하자. 방송과 감소단계에 대한 통신부가처리는 N에 따라 변하지 않는다. 식 (3.17)로부터 확대된 속도증가는 다음과 같이 얻어 진다.

$$S_n^{'} = \frac{\alpha + (1-\alpha)n}{1 + A_0} = \frac{0.00278 + 0.99722n}{1 + T_0/W} = \frac{127.65}{1 + T_0/14.37}$$

T_0을 구하자. 앞의 실례에서 이미 확대된 문제에 대하여 N=3339라는것을 얻었다. 전체 교환단계에서 통보문길이는 $m = 256N^2 / n$ byte=2854/nMB이다.

식 (3.10)으로부터 전체교환에 대한 부가처리는

$$T_{index} = 80\log n + 0.03n^{1.29}\mu \mathrm{s} = 0.00008\mathrm{logn} + 85.62\mathrm{n}^{-0.71}\mathrm{s}.$$

전체 부가처리는

$$T_0 = T_{index} + T_{bcast} + T_{reduce} = 85.62n^{-0.71} + 0.00445\log n + 0.0023.$$

n=128에 대하여 T_0은 2.7s로 평가된다.

확대된 평균부가처리는 $T_0 / W = 2.57 / 14.37 = 0.1914$ 이다.

식 (3.17)로부터 확대된 속도증가는 127.65/1.1914=107인데 부가처리가 무시될 때 얻어 진 127.65속도증가보다 작다.

3. 6. 3. Sun과 Ni의 법칙 : 기억기속박

Xian-He Sun과 Lionel Ni(1993)은 CPU와 기억기용량의 리용을 최대화하기 위하여 Amdahl법칙과 Gustafson법칙을 일반화하는 기억기속박속도증가모형을 전개하였다.

그 방법은 가능한 가장 큰 문제를 푸는것인데 기억공간에 의해 제한된다. 이것 역시 더 높은 속도증가, 더 높은 정확성 그리고 보다 좋은 리용을 제공하는 확대된 작업부하를 요구한다.

기억기속박문제 대규모과학 혹은 공학계산들은 흔히 큰 기억공간을 요구한다. 사실상 병렬콤퓨터에서 CPU속박 혹은 I/O속박보다는 오히려 기억기속박을 많이 리용한다.

이것은 특히 분산기억기를 리용하는 다중콤퓨터체계에서 더욱 그러하다. 매 마디에 붙은 국부기억은 상대적으로 작다. 그러므로 매 마디는 작은 부분문제만을 조종할수 있다. 많은 수의 마디들이 단일한 큰 문제를 풀기 위해 집체적으로 리용될 때 전체 기억용량은 비례적으로 증가한다.

이것은 자료모임의 령역분해를 통해 체계가 확대된 문제를 풀수 있게 한다. 실행시간을 고정시키는것 대신에 문제크기를 더욱 확대함으로써 모든 증가된 기억을 리용할수 있을것이다. 기억기속박모형은 이 원리밑에서 전개되였다.

이 방법은 가능한 가장 큰 문제를 푸는것인데 유효기억용량에 의해서만 제한된다.

실례 3.21. APT성능평가기준프로그람의 기억기요구

표 3-24는 3개의 STA성능평가기준프로그람의 기억요구를 보여 준다. 입력점들은 계산에 필요한 기억기요구만을 보여 준다.

MPPs에서 두배나 많은 기억기가 요구될수 있는데 그것은 통신완충기에서 필요하기

때문이다.

표 3-24 STAP성능평가기준의 프로그람의 기억요구

프로그람	기억기요구(MB)		작업부하 (Mflop)
	단순정확도	배정확도	
APT(min)	0.23	0.46	5
API(norm)	18.35	36.71	1446
APT(max)	3276.80	6553.60	12,100,000
HO(min)	0.15	0.30	21
HO(norm)	50.33	100.66	12,852
HO(max)	3276.80	6553.60	33,263,288
GEN(min)	0.26	0.52	6
GEN(norm)	100	200	5326
GEN(max)	33,957	67,914	4,604,011

최대자료모임이 리용될 때 10Gflops/MPP는 1210s동안에 APT를 처리할수 있고 3326s동안에 FO-PD를 처리할수 있으며 460s동안에 GEN을 처리할수 있다.

대응하는 기억기요구는 13GB만큼 크다.

많은 현대 처리기들은 매개가 약 100Mflop/s를 처리할수 있다.

STAP의 기억기요구를 만족시키기 위하여 10Gflop/s 혹은 100마디, MPP는 마디당 130MB기억을 요구한다.

그러한 기억기용량은 초고속콤퓨터에서만 존재한다. 현재 설치된 대부분의 MPPs는 처리기당 128MB이하의 기억기를 가지고 있다.

기억기속박속도증가 다중콤퓨터에서 전체 기억기용량은 유한개의 마디수에 따라 선형으로 증가한다. M을 단일마디의 기억기용량이라고 하자. n-마디 MPP에서 전체 기억기용량은 nM이다.

기억기속박문제가 주어 지면 한개 마디에서 모든 기억기용량 M을 리용하여 W초에 실행한다고 가정하자(실례로 순차계산론적작업부하는 W이다.). 보통 작업부하는 본질적으로 순차부분과 병렬가능성부분을 가진다. 즉

$$W = \alpha W + (1-\alpha)W$$

n개 마디들이 리용될 때 보다 큰 문제들은 증가된 기억기용량 nM에 의해 풀수 있다. 작업부하의 병렬부분이 $G(n)$배로 확대될수 있다고 가정하자. 즉 확대된 작업부하는 $W = \alpha W + (1-\alpha)G(n)W$ 이다. 인자 $G(n)$은 기억용량이 n배로 증가하는데 따르는 작업부하증가를 반영한다.

기억기속박속도증가는 다음과 같다.

$$S_n^* = \frac{\text{확대된 작업부하에 대한 순차시간}}{\text{확대된 작업부하에 대한 병렬시간}} = \frac{\alpha W + (1-\alpha)G(n)W}{\alpha W + (1-\alpha)G(n)W/n} = \frac{\alpha + (1-\alpha)G(n)}{\alpha + (1-\alpha)G(n)/n} \tag{3.18}$$

순차시간은 nM기억용량을 가진 가상마디를 리용하면 얻어 진다.
모든 부가처리가 고찰될 때 속도증가는 다음과 같다.

$$S_n^{'} = \frac{\alpha W + (1-\alpha)\,G(n)W}{\alpha W + (1-\alpha)G(n)W/n + T_0} = \frac{\alpha + (1-\alpha)G(n)}{\alpha + (1-\alpha)G(n)/n + T_0/W} \tag{3.19}$$

식 (3.19)에는 다음과 같은 세가지 특수경우가 있다.

(1) $G(n) = 1$. 이것은 문제의 크기가 고정된 경우에 대응된다. 따라서 기억기속박속도증가는 Amdahl의 법칙과 같다. 즉 식 (3.19)과 식 (3.15)는 고정작업부하가 주어 졌을 때 등가이다.
(2) $G(n) = n$. 이것은 기억기가 n배 증가될 때 작업부하가 n배 증가하는 경우에 적용된다. 따라서 식 (3.19)는 고정실행시간을 가진 식 (3.17)의 Gustafson법칙과 동일하다.
(3) $G(n) > n$. 이것은 계산론적작업부하가 기억기요구보다 더 빨리 증가하는 경우에 대응된다. 따라서 기억기속박모형 (3.19)는 고정부하속도증가식 (3.15)와 고정시간속도증가식 (3.17)의 두개보다 더 높은 속도증가를 보여 준다.

실례 3.22. 병렬APT의 세가지 속도증가인자비교

병렬 APT프로그람(그림 3−18)이 n마디다중콤퓨터에서 실행된다고 가정하자. 여기서 매 마디는 64MB기억을 가진다. N에 관한 속도증가의 세가지 형태를 도표로 작성하고 모든 통신부가처리를 고찰한다.

APT프로그람은 다음과 같은 추가적인 특징을 가진다.

- 작업부하는 식

$$W = 0.011N^2 + 2.8N + 2.88\text{Mflop}$$
$$W = 0.00016N^2 + 0.015N + 0.04\text{S}$$

과 같이 문제파라메터 N에 따라 변한다. N=256이면 W=1447Mflop 혹은 14.37s 이다.
- 기억요구는 $512N^2$byte이다.
- 방송부가처리 T_{bcast}와 감소부가처리 T_{reduce}는 작업부하가 확대됨에 따라 변하지 않는다. 그것들은 기계크기 n에만 의존한다.
- 전체 교환부가처리 T_{index}는 작업부하가 확대되는데 따라 다음과 같은 률로 증가한다.

$$T_{index} = 80\log n + 0.03n^{1.29}m, \quad m = 256(N/n)^2 \text{ bytes} \tag{3.20}$$

고정작업부하속도증가

실례 3.18로부터 고정작업부하는 시간에 의하여 W=14.37s이다. 순차부분은 α =0.00278 이다.

식 (3.12)로부터 전체 작업부하는 다음과 같다.

$$T_0 = 0.5n^{-0.71} + 0.00445\log n + 0.0023$$

식 (3.15)로부터 고정부하속도증가는 다 음과 같다.

$$\begin{aligned} S_n &= \frac{1}{\alpha + \frac{1-\alpha}{n} + \frac{T_0}{W}} \\ &= \frac{1}{0.00278 + \frac{0.99722}{n} + \frac{0.5n^{-0.71} + 0.00445\log n + 0.0023}{14.37}} \end{aligned} \tag{3.21}$$

고정시간속도증가

원래 작업부하는 여전히 W=14.37s이다. 확대된 작업부하는 αW+(1− α)nW이다. 확대된 작업부하에 대응하는 N은 다음의 식을 풀어 얻는다.

$$0.00016N^2 + 0.015N + 0.04 = \alpha + (1-a)nW$$
$$0.00016N^2 + 0.015N + 0.04 = (000028 + 0.997n)\times 14.37$$

즉 $N = -46.875 + \sqrt{2197 + 8962n}$ 이 다.

식 (3.12), 식 (3.20)와 N값으로부터 전체 부가처리는 다음과 같다.

$$T_0 = \left(0.03 \cdot 10^{-6}\, 256 \cdot \left(-47 + \sqrt{2197 + 8962n}\right)^2\right) n^{-0.71}$$

식 (3.17)로부터 고정시간속도증가는 다음과 같다.

$$S'_n = \frac{\alpha + (1-\alpha)n}{1 + T_0/W} = \frac{0.00278 + 0.99722n}{1 + T_0/14.37} \tag{3.22}$$

고정기억속도증가

n마디체계에 총체적으로 64**n** MB기억기가 있다.

이 큰 기억기에 적재할수 있는 가장 큰 문제크기 N은 다음과 같다.

$$512N^2 = 64n \times 10^6 \Rightarrow N = \sqrt{\frac{64n \times 10^6}{512}} = 353.55\sqrt{n} \qquad (3.23)$$

확대된 작업부하는

$$W^* = 0.00016N^2 + 0.015N + 0.04$$
$$= 20n + 5.3\sqrt{n} + 0.04 = \alpha W + (1-\alpha)G(n)W$$

이다.

α =0.00278, W=14.37로부터 $G(n)$=1.4n+0.37$\sqrt{n}$ 이다.

식 (3.12), 식 (3.20), 식 (3.23)으로부터 전체 부가처리는

$$W^* = 0.00016N^2 + 0.015N + 0.04$$
$$= 20n + 5.3\sqrt{n} + 0.04 = \alpha W + (1-\alpha)G(n)W$$

식 (3.19)로부터 고정기억속도증가는

$$T_0 = 0.03 \times 10^{-6} \times 256 \times (353.55\sqrt{n})^2 n^{-0.71} + 0.00445\log n + 0.0023$$
$$+ 0.96n^{0.29} + 0.00445\log n + 0.0023 \qquad (3.24)$$

세가지 속도증가모형인 식 (3.21), 식 (3.22), 식 (3.24)를 그림 3−9에서 작게 구분하였다.

$G(n)$=1.4n+0.37$\sqrt{n}$ >n이기때문에 고정기억속도증가는 고정시간과 고정작업부하속도증가보다 더 좋다.

속도증가법칙의 개괄 목적은 고정작업부하문제의 실행시간을 줄이는것이라면 체계의 확대가능성은 고정부하속도증가로 정의될수 있으며 일반화될수 있다.

그러나 병렬체계는 문제크기(작업부하)를 기계크기에 따라 Amdahl법칙[식(3.15)]에 의해 결정된다.

가장 좋은 확대된 속도증가를 얻기 위하여서는 Sun과 Ni법칙[식(3.19)]에 의해 결정되는 기억기속박확대를 리용하여야 한다.

문제를 확대하는것은 보다 큰 기계크기가 쓰인다고 해도 더 큰 실행시간을 요구할수 있다. 고정부하와 고정기억확대사이 중간물은 Gustafson의 고정기억확대(식(3.17))인데 어떤 부가처리를 제외하고 원래작업부하의 순차실행시간을 초과하지 않을 확대된 작업부하의 병렬실행시간을 담보한다.

모든 경우에 도달가능한 속도증가는 순차병목문제 α와 평균부가처리 T_0/W에 의해 제한된다. 여기서 T_0은 확대된 작업부하에 대한 병렬프로그람의 전체 부가처리이고 W는 원래 작업부하이다(즉 확대되지 않은).

순차병목문제 α는 특히 고정부하리용에 대해 중요하다.

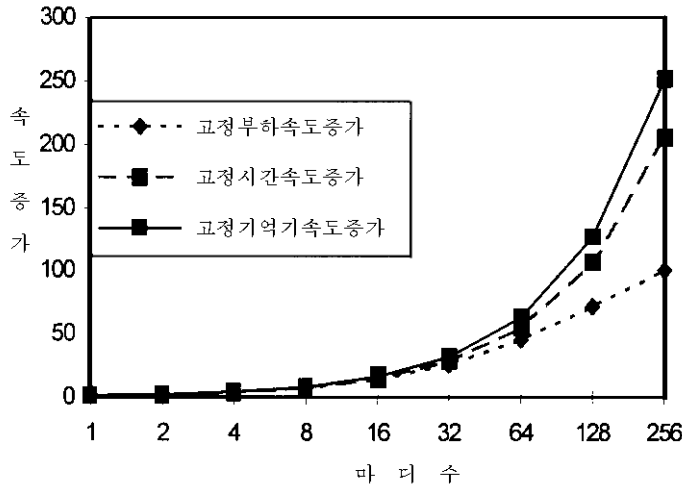



그림 3-9. 병렬 APT프로그람에 대한 세가지 속도증가모형비교

3. 6. 4. 성능고정모형

몇가지 보충적인 척도를 확대가능성을 특징 짓기 위해 정의한다. 그가운데서 3가지를 아래에서 토론한다. 효률성, 처리기당 속도, 리용성을 각각 그대로 유지하면서 병렬체계가 얼마나 잘 확대되는가를 특징 지으려고 한다. 이 절에서 체계라는 용어는 리용(작업부하)과 병렬콤퓨터가동환경을 포함하는 쌍을 가리킨다. 큰 체계를 리용하는것은 품이 많이 든다. 세가지 척도는 먼저 작은 체계에서 확대가능성함수를 유도한 다음 그것을 큰 체계에서의 성능예측에 리용하도록 한다.

효률성고정 Grama등은 [278]체계의 확대가능성을 특징 짓는 효률성고정척도를 제기하였다.

체계의 효률성은 작업부하 W와 기계크기 n 즉 $E=f(W, n)$의 함수로서 표현할수 있다. 효률성을 어떤 상수(실례로 50%)로 고정하고 작업부하 W에 대한 효률성방정식을 푼다면 결과함수를 체계의 **효률성고정함수**라고 부른다.

효률성고정이 작으면 작을수록 작업부하는 같은 효률성을 보장하면서 기계크기증가에 관계있는 증가를 더 적게 요구하며 따라서 체계의 확대가능성이 더 좋게 된다.

실례 3.23. 두개의 행렬곱하기도식

두개 행렬곱하기도식 A, B가 주어 지면 둘 다 같은 수의 처리소자를 가진다. 병렬실행시간과 이 체계들의 효률성을 표 3-25에 제시한다.

두개 $N \times N$행렬들을 곱한 순차시간은 약 $T_1 = cN^3$이고 효률성은 $E = T_1/(nT_n)$으로 정의된다는것에 주목한다.

표 3-25 두개의 행렬곱하기도식의 병렬시간과 효률성

도식	병렬실행시간 T_n	효률성 E
A	$cN^3/n+bN^2/\sqrt{n}$	$E(A)=\frac{1}{1+(b\sqrt{n})/(cN)}$
B	$2cN^3/n+bN^2/(2\sqrt{n})$	$E(B)=\frac{1}{2+(b\sqrt{n})/(2cN)}$

두개의 병렬시간표현에서 첫번째 항은 계산시간을 가리키고 두번째 항은 통신부가처리를 가리킨다. 두 체계사이차이는 체계 B는 계산시간을 두배로 한것으로 하여 체계 A의 부가처리를 절반만 가진다.

E=1/3과 E=1/4일 때 어느 체계가 더 확대가능한가를 보자.

(1) $N\times N$행렬곱하기문제의 작업부하는 근사하게 $W=N^3$이다. 체계 A에 대하여 E=1/3은 $(b\sqrt{n})/(cN)=2$이라는것을 의미하며 따라서 효률성고정함수는 $W(A)=(b/2c))^3n^{1.5}$이다. 류사하게 체계 B에 대하여 효률성고정함수는 $W(A)=(b/2c))^3n^{1.5}$로 계산된다. 따라서 효률성고정에 의해 두 체계들은 똑같이 확대가능하다.

(2) 체계 A에 대하여 E=1/4는 $(b\sqrt{n})/(cN)=3$이라는것을 의미하며 따라서 효률성고정함수는 $W(A)=(b/(3c))^3n^{1.5}$이다. 류사하게 체계 B에 대하여 효률성고정함수는 $W(A)=(b/4c))^3n^{1.5}$이다. 따라서 체계 B는 보다 작은 효률성고정을 가지며 체계 A보다 더 확대가능하다.

효률성고정척도는 기계크기가 증가하는데 따라 요구되는 작업부하증가률을 예측하기 위한 쓸모 있는 도구를 제공한다. 실례로 실례 3.23에서 효률성고정함수들은 상수효률성을 유지하기 위하여 기계크기를 두배로 하면 작업부하를 2.82인자로 증가시킬것을 요구한다는것을 말하여 준다.

그러나 실례 3.14는 효률성고정척도를 주의깊게 사용하여야 한다는것을 보여 준다. 효률성고정에 의해 더 확대가능한 체계는 반드시 더 빨리 동작되지 않는다.

속도고정 Sun and Rover[394]는 체계확대가능성을 특징 짓기 위해 속도고정개념을 제기하였다.

기본개념은 효률성고정과 류사하다. 상수효률성을 유지하는 대신에 기계크기와 문제크기 두개를 동시에 확대하면서 속도를 보존할수 있다.

리용고정 리상적인 확대가능성척도는 다음과 같은 두가지 성질을 가져야 한다.

(1) 효률성고정 혹은 속도고정과 같이 그것은 기계크기증가에 따라 요구되는 작업부하증가률을 예측한다.

(2) 그것은 실행시간과 일치한다. 즉 더 확대가능한 체계는 언제나 보다 짧은 실행시간을 가진다. 그러한 성질들을 가진 단일확대가능한 척도가 있는가? 대답은 있다는것이며 아래에서 설명한다.

정의 3.1 표 3−20으로부터 병렬체계리용을 $U=P_n/(n \cdot P_{peak})=W/(n \cdot T \cdot P_{peak})$로 정의한다는것을 상기하자. 리용을 어떤 상수(실례로 25%)로 고정하고 작업부하 W에 대한 리용방정식을 풀면 결과함수를 체계의 **리용고정함수**라고 부른다.

효률성고정과 같이 리용고정척도는 같은 리용을 유지하면서 기계크기증가에 관계하는 요구되는 작업부하증가률을 예측할수 있다. 작은 리용고정을 가진 체계는 큰 리용고정을 가진 체계보다 더 확대가능하다.

더우기 아주 합리적인 조건밑에서 보다 작은 리용고정(따라서 보다 확대가능한)은 언제나 보다 짧은 실행시간을 가진다는것을 보여 줄수 있다. 이 결과의 자세한 유도는 련습으로 남긴다(문제 3.13).

실례 3.24. 병렬행렬곱하기도식의 리용고정

실례 3.23을 다시 고찰하자. 체계 A의 리용은

$$U=\frac{W}{nT_n(A)P_{peak}}=\frac{N^3}{n(cN^3/n+bN^2/\sqrt{n})\,p_{peak}}$$
$$=\frac{1/(cP_{peak})}{1+(b\sqrt{n})/(cN)}$$

고정리용 $U=1/(4cP_{peak})$에 대하여 다음의 식이 성립한다.

$$b\sqrt{n}/(cN)=3\Rightarrow W(A)=(b/(3c))^3\cdot n^{1.5}$$

류사하게 체계 B의 리용은

$$U=\frac{W}{nT_n(B)P_{peak}}=\frac{N^3}{n(2cN^3/n+bN^2/2\sqrt{n})\,p_{peak}}$$
$$=\frac{1/(2cP_{peak})}{1+(b\sqrt{n})/(4cN)}$$

같은 고정리용 $U=1/(4cP_{peak})$에 대하여

$b\sqrt{n}/(4cN)=1\Rightarrow W(B)=[b/(4c)]^3\cdot n^{1.5}$

$W(B)<W(A)$ 로부터 체계 B가 더 확대가능하다는것을 알수 있다.

실례 3.25. 리용고정에 대한 확대가능성

실례 3.14로부터 b=4c, N>2$\sqrt{n}$일 때 체계 A가 체계 B보다 더 빨리 동작한다는것을 알수 있다. 이것을 체계 B가 더 확대가능하다는 실례 3.24의 결론과 어떻게 조화시키겠는가?

즉 b=4c, N>2일 때 체계 A의 리용은 다음과 같다.

$$U = \frac{1/(cP\ peak)}{1+(b\sqrt{n})/(cN)} > \frac{1}{3cP_{peak}}$$

역시 체계 B에 대한 같은 결론이 성립한다. 이것은 실례 3.24에서의 고정리용 U=1/(4cP_{peak})과 모순된다. 이제 가정을 고정리용 U=1/(4cP_{peak})로 변환시키자.

체계 A에 대하여

$$b\sqrt{n}/(c^{a}N) = 1.4 \Rightarrow \text{¤} W(A) = (b/(1.4c))^{3} \cdot n^{1.5}$$

체계 B에 대하여

$$b\sqrt{n}/(4cN) = 0.2 \Rightarrow W(B) = (b/(0.8c))^{3} \cdot n^{1.5}$$

그러면 체계 B는 보다 작은 리용고정을 가지며 따라서 더 확대가능하다.

이 실례는 체계가 다른것보다 언제나 반드시 더 확대가능하지 않는다는것을 보여 준다. 그것은 선택된 리용수준에 의존한다.

우리는 리용값을 임의로 선택할수 없다. 체계의 응용프로그람은 언제나 100%보다 작으며 흔히 그것보다 훨씬 작다.

실례 3.28에서의 두개 행렬곱하기체계에 대하여 체계 A의 리용은 1/(cP_{peak})를 초과할수 없으며 체계 B의 리용은 1/(2cP_{peak})를 초과할수 없다.

3. 7. 참고문헌주해와 련습문제

Bhuyan와 Zhang이 편집한 지도안 (tutorial)은 다중처리소자성능의 측정과 평가를 취급한다.

Bell[67]은 사용자와 설계자의 전망으로부터 확대가능한 성능을 특징 지었다. Hockney의 문헌들과 관련문헌들[313~318]은 병렬성능척도에 대한 깊은 토론들을 포함하고 있다.

Karp와 관련문헌들은[362~364] 여러가지 병렬 Fortran변종을 비교하고 병렬리용성능을 어떻게 평가하는가를 지적하였다.

여러가지 성능척도사이의 관계는 Sahni와 Thanrantri에 의해 토론되였다.

Amdahl법칙은 먼저 1967[29]에 제기되였다.

Gustafson[286]은 보다 큰 체계가 쓰일 때 문제크기를 확대함으로써 고정실행시간도달방법을 제기하였다.

Gustafson과 Snell은 최근에 확대가능한 문제크기와 시간인 HINT라고 부르는 간단한 성능평가기준을 제기하였다.

Sun과 Ni[596]은 기억기속박속도증가모형을 전개하였다.

효률성고정개념은 Kymar와 관련문헌 [278,383,384]에 의해 전개되였다. Isospeed개념은 Sun과 관련문헌 [594~595]에 의해 취급되였다.

실지 프로그람에서 유효병렬성은 Blume 등 [92],Wall[632], Worley[650]와 Hwang과 Xu[333]에 의해 해석된다. 확대가능한 성능법칙들은 역시 Hwang[327], Almasi와 Gottlieb[26], Lenoski와 Weber[406]에서, NAS성능평가기준결과들은 때때로 Bailey와 관련문헌 [54, 525]에서 보고되였으며 Dongarra[209]에 의해 LINPACK성능평가기준이 보고되였다.

병렬체계성능은 역시 Lenoski 등의 [405, 406], Agarwal 등의 [12], Stenstrom 등의 [585], Wang a Hwang[635], Athas와 Seitz[45]에 의해 보고되였다.

제시된 STAP실례들은 [655, 656, 657]에서 보고된 Xu와 Hwang문헌에 기초하였다.

우리는 통신과 병렬성관리에서 부가처리를 평가하는 량적인 방법을 전개하였다.

이 결과들은 병렬체계성능초기예측에 적용되였다.

여러가지 성능해석과 평가도구는 역시 [189, 301, 460, 514, 517564]에서 쓸모 있다.

문 제

문제 3.1. 사용자에게 쓸모 있는 콤퓨터성능을 검사하는 GINPACK를 리용한다. 기계가 병렬인가 아닌가는 중요하지 않다. 물론 병렬성능을 측정하는것은 더 재미 있을것이다.

(1) 그것의 R_{max}, N_{max}, $N_{1/2}$, R_{peak}값을 찾으시오.

(2) 체계리용은 무엇인가?

(3) 최대성능/비용률은 얼마인가?

(4) 지속성능/비용률은 얼마인가?

문제 3.2. 사용자가 좋아 하는 콤퓨터를 검사하는 STREAM성능평가기준을 리용하여 그것의 TRIAD대역너비와 기계평형을 찾으시오. 얻어 진 TRIAD대역너비와 콤퓨터최대하드웨어기억기대역너비를 비교하시오.

문제 3.3. 지금 콤퓨터가 계산(MIPS와 Mflops), 자료이동(기억과 디스크, MB/s), 체계호출, 프로쎄스관리, 통신과 동기화 등을 얼마나 잘 지원하는가를 충분히 측정하는 포괄적인 마이크로성능평가기준은 없다.

(1) 아주 쓸모 있고 도전적인 클라스계획은 그러한 한조의 마이크로성능평가기준을 개발하고 그것들을 이식성묶음에 통합하여 WWW의 공동령역에 넣는것이다.

(2) 클라스계획성능평가기준을 공동령역성능평가기준들 즉 LINPACK, STRAM, LMBENCH, TARKBENCH와 비교하시오.

문제 3.4. 실례 3.11의 류형에 따라 점대점통신지연시간을 측정하는 Ping−Pong도식을 실현하기 위한 SPMD프로그람을 쓰시오.

문제 3.5. 집중통신(실례로 전체 교환)시간을 측정하는 방법을 제기하시오. 그 방법은 3.4.3절에서 토론한 방법들과 차이가 있다. 새로운 전체 교환알고리듬의 개요를 말하고 시간복잡성과 완성효률성의 견지에서 개선된것을 확인하시오.

문제 3.6. n마디 SP2에서 그림 3−8에 있는 병렬 APT알고리듬의 실행시간 T_n을 유도하시오. $n \le 256$일 때

$$T_n = T_{comp} + T_{par} + T_{inteaact}$$
$$= \frac{14.33}{n} + 0.5n^{-0.71} + 0.00445\log n + 0.0423$$

전체 작업부하는 SP2마디에서 W=1447Mflop 혹은 14.37s이다.
그림 3−8에서 병렬 APT알고리듬의 축소단계에서 통신된 byte수는 약 0.1MB이다.

문제 3.7. 식 (1.4)를 리용하여 1.5.2절에서의 균형설계에 대한 50%규칙을 유도하시오.

(1) 매 부하불균형, 출발시간, 통신대역너비, 병렬성부가처리인자들에 대하여 부등식을 유도하시오.
(2) 표 1.9와 표 1.10을 유도하시오.

문제 3.8. 실례 3.16을 리용하여 다음질문에 대답하시오.

(1) 기초프로그람은 흔히 수값프로그람보다 더 작은 병렬성을 가진다. Gcc프로그람이 왜 sgefu프로그람보다 더 작은 평균병렬성을 가지는가 하는 리유를 말하시오.
(2) 기초프로그람을 더 효률적으로 동작시키기 위하여 병렬콤퓨터의 성능을 개선하는 방도를 제기하시오. 구성방식지원, 프로그람재구조화, 콤파일러최량화 등과 같은 모든 면에서 잘 완성할수 있다.

문제 3.9. 두개의 $N \times N$행렬들을 곱한 순차시간은 약 $T_1 = cN^3$이다. 여기서 C는 상수**.** 병렬행렬곱하기도식은 n−마디콤퓨터에서 병렬실행시간 $T_n = cN^3/n + bN^2/\sqrt{n}$를 가진다. 여기서 b는 다른 상수이다.

(1) 첫번째 항은 계산시간을 의미하고 두번째 항은 통신부가처리를 의미한다.
(2) 고정작업부하속도증가를 찾고 결과에 대하여 해설하시오.
(3) 고정시간속도증가를 찾고 결과에 대하여 해설하시오.
(4) 기억기속박속도증가를 찾고 결과에 대하여 해설하시오.

문제 3.10. A, B 라고 부르는 콤퓨터에서 다음과 같은 응용프로그람실행시간을 비교하시오. 응용프로그람은 8192−점 FFT(고속푸리에변환)과 독립인 2048로 이루어 진 Doppler처리이다.

사용자는 기계 A에서 50s의 실행시간을 측정하고 B에서는 30s를 측정한다고 가정하자. 모든 부가처리시간을 무시한다.

(1) 분당 수행되는 FFT수로서 기계 B의 속도는 얼마인가? B는 A에 대해 얼마나 빠른가?

(2) 이제 사용자가 화상실감묘사를 수행하는것이 필요하다고 가정하자. 사용자가 두 기계에서 실감묘사코드를 실행하는데 기계 A는 화상프렘을 실감묘사하는데 6분 필요하고 기계 B는 10분 필요하다고 한다. 전형적인 작업부하혼합은 매 화상실감묘사에 대하여 40도플러처리리용이다. 두 기계에서 실행시간은 얼마인가? 어느 기계가 더 빠르며 얼마나 더 빠른가?

(3) 작업부하혼합을 매 화상실감묘사에 대하여 10도플러처리응용으로 변화시킬때 (2)부분을 반복하시오.

문제 3.11. 작업부하 W=1, 령순차병목($\alpha = 0$)을 가진 프로그람을 고찰하자. 다음과 같은 매 부가처리가정에 대하여 식 (3.17)에서의 고정시간속도증가인자 $S_n^{'}$을 평가하시오.

(1) 부가처리 $T_0 = O(n_{0.5})$에 대한 확대된 속도증가의 big−O표현은 무엇인가?

(2) 병렬계산리용에서 superlinear인 속도증가를 가지는 가능한 리유는 무엇인가?

(3) 부가처리 T_0=O(log$n$)에 대해 (1)부분을 반복하고 결과를 해석하시오.

(4) 부가처리가 상수보다 크면 선형속도증가를 얻는것이 가능하다.

문제 3.12. 실례 3.22를 리용하여 다음질문에 대답하시오.

(1) 고정효률성 1/5을 가정하고 SP2에서 동작하는 병렬APT성능평가기준의 리용고정함수를 유도하시오.

(2) 고정효률성 1/10을 가정하고 SP2에서 동작하는 병렬APT성능평가기준의 리용고정함수를 유도하시오.

문제 3.13. 두개의 처리기 A, B가 주어 지고 그것들의 리용고정함수를 각각 $W(A)$, $W(B)$라고 하자.

$W(A) < W(B) \Rightarrow T_n(A) < T_n(B)$이기 위한 필요충분조건을 유도하시오. 즉 보다 확대된 체계가 더 빠르다.

문제 3.14. 이 문제는 Sahni과 Thanvantri문헌에 기초한다[524]. 실례 3.23에서의 두개 행렬곱하기도식을 고찰하자. 고정효률성 E=0.25를 가정하고 체계 B는 체계 A보다 효률성고정척도에 의해 더 확대가능하다. 모든 값 b, c>0에 대하여 b=4c로 가정한다. $W(A)=1.33^3 n^{1.5}$, $W(B)=n^{1.5}$

(1) 우와 같은 조건에서 B는 A에 비해 얼마나 더 확대가능한가?

(2) 표 3−25에서 병렬실행시간을 보시오. $N>2\sqrt{n}$일 때 비률 (2)T_n/T_n(1)에 대한 식을 유도하시오.

(3) 체계 B는 체계 A보다 더 느릴수 있는가? 그 리유를 설명하시오.

제2편. 허 용 기 술

이 편은 4개 장으로 이루어 져 있다. 4, 5, 6장은 처리기, 기억기 그리고 호상접속기술들에 대하여 서술하고 7장은 병렬성, 호상작용 그리고 통신에 대한 쏘프트웨어지원에 대해 서술한다.

4장에서는 구성방식, 기술과 상품화된 극소형처리기들의 응용들을 소개한다. 하드웨어와 쏘프트웨어개발추세에 대한 개괄부터 시작해서 CISC, RISC, post RISC와 미래의 가능한 구성방식을 서술한다. 특히 초스칼라, 초관흐름, 분리형CISC/RISC VLIW와 현재극소형처리기들과 매몰형처리기들의 다매체확장을 연구한다. 또한 큰 체계들의 제작블로크로서 이 극소형처리기들을 리용하는데서 다중처리지원을 조사한다.

5장에서는 주로 분산기억기구성방식을 서술한다. 계층기억기기술로부터 출발해서 MESI snoopy와 등록부에 기초한 규약을 포함하는 캐쉬일관성규약을 고찰하고 공유기억기일치성모형을 소개한다. 또한 UMA, NORMA, CC-UNMA, COMA 그리고 DSM모형을 고찰한다. 분산캐쉬, 자료미리꺼내기(prefetching), 완화형(relaxed)기억기일치성의 성능평가기준과 긴 기억지연시간을 은폐하는 다중스레드처리기리용에 따르는 지연시간허용기술을 제시한다. DSM의 하드웨어지원과 쏘프트웨어도구가 제시되며 자세한것은 8, 10장에서 보기로 한다.

6장에서는 기가비트망과 절환되는 호상접속들이 고찰된다. 망형태, 위상수학, 기능성들과 성능척도를 포함하여 호상접속망기초부터 시작한다. 점대점호상접속외에 또한 다중처리기모선들과 크로스바(crossbar), 다계단절환(switch)들을 고찰한다. 취급되는 고속망은 기가비트 Ethernet, Fibre통로, FDDI고리, Myrinet, HiPPI, SCI, ATM기술들을 포함하며 기가비트망을 리용하는 실례들을 보여 준다. 특히 기가비트 Ethernet, IEEE SCI표준, ATM절환과 인터네트망을 자세히 고찰하며 여러가지 호상접속, 망 그리고 절환들의 량적인 비교가 주어 진다.

7장에서는 독자들에게 개념과 스레드, 동기화, 다중처리기 혹은 다중콤퓨터에서 유효통신의 쏘프트웨어지원을 리해하는데 필요한것들을 제시한다. Solaris에서 스레드, 동기화기구 PVM, MPI, BCL, SP2에서 MPL, TCP/IP, UDP규약, 관, 유닉스환경에서 주로 고속통신을 위한 소케트(socket)대면부를 설명한다.

제4장. 제작블로크: 극소형처리기

상품화된 극소형처리기들은 확대가능한 체계들과 클라스터콤퓨터체계들을 제작하는데 쓰이고 있다.

이 장에서는 극소형처리기의 체계개발추세와 구성방식을 고찰한다.

먼저 하드웨어, 쏘프트웨어병렬처리기들과 콤퓨터클라스터의 리용들에서의 최근 발전으로부터 출발해서 그다음 기본적인 VLSI기술, 극소형구성방식설계, 그라픽스와 다매체확장을 고찰한다.

특히 명령준위병렬성, 순서가 없는(out-of-order) 동적이고 투기적인 실행, 기억기지연시간감소, 다중처리지원, post-RISC특징 그리고 미래의 극소형처리기에서 구성방식을 개발하기 위한 새로운 기구와 구성방식지원과 같은 성능특징에 관심을 두고 고찰한다.

4. 1. 체계개발추세

고성능콤퓨터들은 일반목적과 일반구성방식을 가진 확대가능한 체계들로 발전하고 있다. 이 절에서는 이 체계들에서 하드웨어, 쏘프트웨어리용의 개발추세를 고찰한다. 콤퓨터체계의 중요한 리용은 정보의 처리, 정보기억, 전송을 포함한다. 현대 콤퓨터기술에서 몇가지 부족점을 보면 다음과 같다.

- 자료전송속도는 처리능력보다 떨어 진다. 즉 콤퓨터는 처리기속도에서 아주 빠르지만 처리기와 기억기, 디스크, 망사이정보를 이동하는데서는 느리다.
- 쏘프트웨어는 하드웨어보다 떨어 진다. 쏘프트웨어는 사용자들이 하드웨어용량을 충분히 리용할수 없게 한다.

체계의 기본비효률성은 쏘프트웨어비효률성에 기인된다. 나아가서 체계쏘프트웨어와 응용쏘프트웨어 두개는 언제나 하드웨어보다 한세대 뒤떨어 져 있다.

4. 1. 1. 하드웨어의 발전

처리기들 병렬콤퓨터체계의 구성요소들가운데서 처리기는 가장 빠른 기술적진보를 이룩하였다.

그림 4-1에 Intel 80×86극소형처리기계렬에 대한 자료가 제시되였는데 그것을 보면 시간에 따라 성능개선이 지수적이라는것을 보여 준다. 바로 17년이 지나서 밀도(즉 극소형처리기당 3극소자수)는 거의 200배, 박자속도는 31배, 최대속도는 900배 증가하였다.

이 수자들은 년간 평균증가률이 밀도에서 36%, 박자속도에서 22%, Mi/s속도에서는 49%라는것을 반영한다. 속도증가의 크기는 박자속도대신에 처리기구성방식에서의 발전으로부터 이루어 졌다. 4.3절에서 현대 극소형처리기의 구성방식발전을 고찰한다.

Moore법칙 지수적인 증가현상은 Intel cofounder Gordon Moore에 의하여 이미 1979년에 관찰되였다. Moore법칙은 3가지로 해석된다.

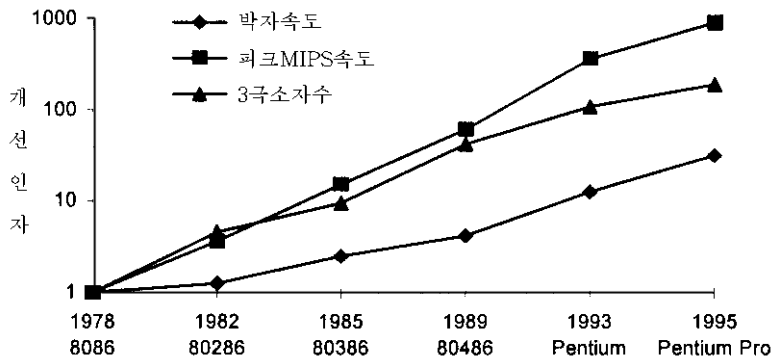



그림 4-1. Intel 극소형처리기의 개선추세

- 미소소편에서 3극소자수는 소편가격이 갈다고 하면 대략 18~24달마다 두배 증가한다.
- 극소형처리기속도는 처리기값이 갈다고 하면 대략 8~24달마다 두배 증가한다.
- 미소소편의 가격은 같은 처리기속도 혹은 소편에서 기억기용량이 갈다고 하면 18~24달마다 대략 48% 떨어 진다.

기억기의 발전 기억기는 처리기만큼 중요한 구성요소이다. 그러나 기억기기술의 발전은 IBM PC계렬에 대하여 그림 4-2에서 보는바와 같이 처리기발전에 보조를 맞추지 못하였다. PC/XT와 비교하면 Pentium Pro PC는 900배 이상으로 처리기속도를 증가시켰으나 주기억기용량은 64배, 하드디스크용량은 85배 증가되였으며 기억기호출시간은 기껏 10배로 증가되였다.

더우기 곡선은 다른 3개 파라메터들이 여전히 지수적으로 증가하는 동안에 주기억기호출시간은 386 based PC에서부터 선형으로 증가한다는것을 보여 주었다.

RAM접근시간은 처리기속도보다 아주 느리게 증가한다.

그러나 어떤 체계들은 느린 기억기호출시간에도 도달할수 없다.

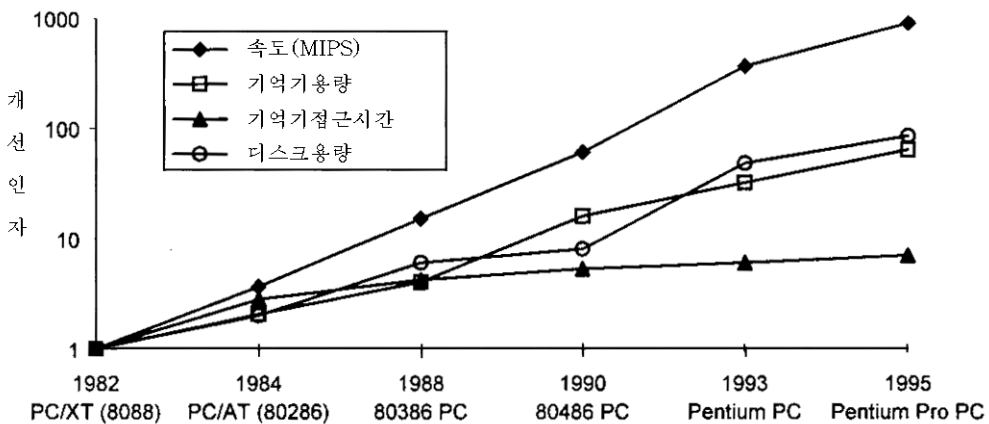



그림 4-2. IBM PC세대에서 하드웨어발전

실례로 DEC 8000봉사기에서 기억기소편은 60ns의 호출시간을 가진다 그러나 기억기지연시간은 처리기가 주기억기부터 단어를 읽는 시간이며 250ns로써 훨씬 높다.

다른 체계들에서 형편은 더욱 나쁘다.

기억기지연시간은 RAM소편에서 60~80ns로 하면 600ns만큼 높아 진다.

디스크와 테프단 반도체소편으로 만든 주기억기는 콤퓨터체계에서 최초의 기억장치이다. 두번째, 세번째 기억장치는 각각 주로 자기디스크, 테프로 만들었다. 그것들은 또한 대용량기억장치로 알려 져 있다.

대용량기억장치는 기계적운동을 포함하기때문에 그것들의 속도는 기억기보다 훨씬 느리다. 지금은 접근시간과 기억기대역너비가 디스크보다 크기에서 2~3준위 더 좋고 테프단 보다는 3~7준위 더 좋다.

더우기 RAMs속도는 디스크장치, 테프단보다 높은 비률로 증가한다. 디스크/테프단은 흔히 I/O중심응용에 대해서는 병목문제이다.

통신부분체계 Intel MPPs의 통신성능을 그림 4-3의 처리속도와 비교한다. 여기서 처리기속도는 Intel MPPs에서 쓰인 단일처리기의 최대Mflop/s속도이다. 대역너비와 시동지연시간은 점대점통보문을 보내는것에 대한 가장 좋은 평가기준이다.

다음과 같은 3가지 관찰들을 그림 4-3에서부터 얻을수 있다.

첫째로, 모든 3개 성능척도는 지수적으로 증가한다.

둘째로, 두개 통신성능척도가운데서 대역너비는 시동지연시간보다 더 빨리 증가한다.

셋째로, 처리기속도(즉 계산용량)는 통신용량보다 훨씬 높은 비률로 증가한다. 1996년까지 10년동안에 Intel MPPs의 처리기속도는 5000배 증가하였으며 대역너비는 760배 증가하였지만 시동지연시간은 단지 86배만 증가하였다.

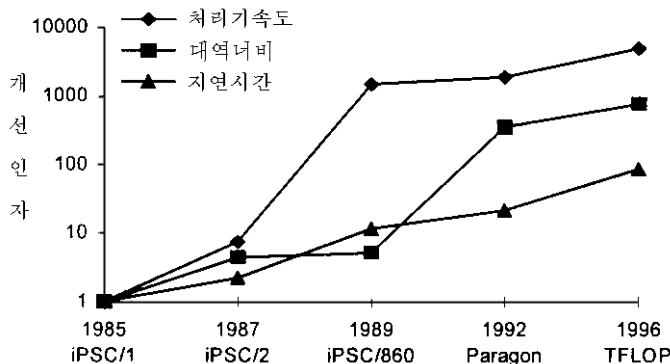



그림 4-3. Intel MPP계렬의 계산과 통신용량비교

실례 4.1. Cray Intel초고속콤퓨터의 발전

두 초고속콤퓨터계렬의 발전추세를 그림 4-4에서 보여 주었다. 하나는 proprietory처리기를 리용하고 다른것은 Intel극소형처리기를 리용한것이다.

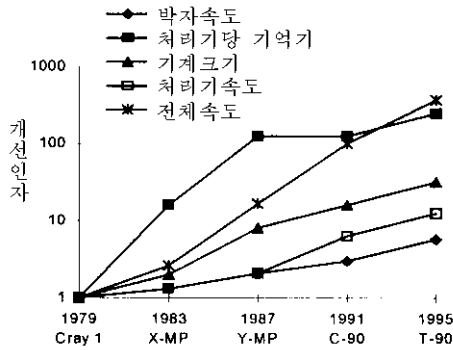
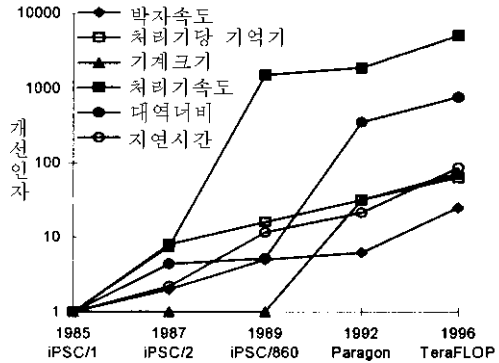



그림 4-4. Cray 초고속콤퓨터와 Intel 병렬처리기의 발전추세

표 4-1은 모든 자료점들을 종합한다. Cray처리기최대속도는 16년동안에 주로 박자속도에서 12.5배 증가하였다.

10년기간 Intel극소형처리기는 속도에서 5000배 증가하였는데 그 가운데서 25배만이 아주 빠른 박자속도로부터 나오며 나머지 200배는 극소형구성방식의 발전에서 나온다.

같은 주기동안에 Intel MPPs에 대한 한통로(one-way)점대점통신대역너비는 760배 증가하였으며 지연시간은 86배로 증가하였다. Cray초고속콤퓨터는 주기억기로서 고속 RAMs을 리용하며 주문설계화된 크로스바(custom-designed crossbar)는 높은 대역너비와 낮은 통신지연시간을 제공한다. 결과 Cray초고속콤퓨터에서 동작하는 응용프로그람은 흔히 다른 MPPs(1%~30%)에서보다 더 높은 리용을 가진다(15%~45%).

표 4-1 Cray초고속콤퓨터와 Intel MPP계렬의 발전

판매자	콤퓨터	년도	박자	기억기	속도	대역너비	시동시간
Cray	Cray 1	1979	80	1	160	알려 지지 않음	알려 지지 않음
	X-MP	1983	105	2	210	알려 지지 않음	알려 지지 않음
	Y-MP	1987	166	256	333	알려 지지 않음	알려 지지 않음
	C90	1991	238	256	1000	9444	0.1
	T90	1995	454	512	2000	알려 지지 않음	알려 지지 않음
Intel	IPSC/1	1985	8	0.5-4.5	0.04	0.5	862
	IPC/2	1987	16	4-16	0.3	2.2	390
	IPSC/860	1989	40	8	60	2.6	75
	Paragon	1992	50	16-128	75	175	40
	TFLOPS	1996	200	16-128	200	380	10

4. 1. 2. 쏘프트웨어의 발전

순차쏘프트웨어의 상태에 비하면 병렬쏘프트웨어의 발전은 낮은 상태에 있다.

병렬기계는 사용자들이 2장에서 고찰한 많은 새로운 론점에 직면하고 있기때문에 프로그람화하기가 힘들다.

더우기 병렬프로그람작성을 지원하는 좋은 쏘프트웨어도구가 결핍되여 있다. 많은 도구들은 병렬체계구성방식의 빠른 변화로 하여 그것이 완성된후 곧 낡아 진다.

결과적으로 많은 병렬응용프로그람들은 순차Fortran 혹은 C와 서고함수 혹은 처리관리와 호상작용을 위한 콤파일러명령을 리용하여 발전하였다.

일반적인 경우에 쏘프트웨어도구(혹은 간단히 도구)라는 용어는 체계가 아래와 같이 말단사용자에게 제공하는 쏘프트웨어리용을 의미한다.

- 조작체계리용
- 응용부분체계와 미들웨어(middleware)(실례로 자료기지와 단위업무처리모니터)
- 프로그람작성언어와 콤파일러
- 서고들(실례로 기본계산, 통신, 다중스레드작성을 위한)
- 프로필링, 오유수정, 가시화리용

고성능콤퓨터들은 DSMS, MPP, 클라스터로 지향하고 있고 병렬쏘프트웨어의 위기는 더 심각해 지고 있다.

그러나 아래와 같은 몇가지 문제는 눈에 뜨이게 발전하였다.

- 병렬쏘프트웨어에 무엇이 필요되는가를 잘 알고 있다. 관건적인 문제들과 론점들이 확인되고 유효풀이가 명백해 지고 있다.
- 각이한 플래트홈들가운데서 이식성 있고 몇개의 기계세대에 대하여 확대가능한 구성방식의존성도구를 개발해야 한다는 의견이 일치되고 있다. 이 의존성원리는 어떤 성능을 떨구더라도 유지되여야 한다.
- 오히려 공개령역에 있는 열린 표준도구들을 개발하여야 한다. 그러한 도구들의 실례들로는 다중스레드화를 위한 Pthread, 자료-병렬프로그람작성을 위한 HPF, 통보문넘기기를 위한 MPI, PVM을 포함한다.
- 체계쏘프트웨어와 응용쏘프트웨어판매자들은 자기들의 생산품의 병렬차원수를 발전시키고 있다. 실례로 모든 주요자료기지판매자들은(IBM, Oracle, Sybase, Informx) 지금 병렬자료기지를 준비하고 있다.

4. 1. 3. 응용프로그람의 발전

응용령역에서 병렬쏘프트웨어는 역시 훨씬 뒤떨어 져 있다.

특히 상업적응용을 위하여 개발된 병렬프로그람은 거의 없다. 병렬응용프로그람을 개발하는것은 시간이 드는 과제이다. 그러므로 성능, 이식성, 확대가능성은 프로그람발전

속에서 고찰되여야 한다.

몇가지 주목할만한 발전들을 언급하자.

병렬파라다임 몇개의 순환파라다임은 계산–호상작용(compute−interact), 동기반복, 비동기반복, 관흐름, 분할획득(divide−conquer), process farm(master/slave)과 process pool와 같은 병렬알고리듬을 구조화하는데 결합된다.

병렬알고리듬 과거 10년은 병렬알고리듬이 폭발적으로 축적되였다. 많은 새로운 병렬알고리듬들이 매일 발견되였다. 유감스럽게도 대부분의 그러한 병렬알고리듬들은 비현실적인 PRAM모형 혹은 그것의 변종들에 기초하고 있는데 중요하고 상세한것들은 등한시하였다. 쓸모 있고 현실적인 병렬응용프로그람을 개발하기 위하여서는 이 알고리듬들이 보다 실천적인 BSP모형 혹은 위상모형에로 넘어 가야 한다.

병렬표준 응용프로그람들은 고수준, 표준언어(실례로 Fortran 혹은 C)와 표준서고함수(실례로 MPI 혹은 PVM) 혹은 콤파일러명령(실례로 HPF)을 써서 코드화되여야 한다.

독립성 응용프로그람은 한번 개발되면 각이한 플래트홈과 각이한 기계크기에서 적게 수정되면서 효률적으로 실행할수 있어야 한다. 코드는 호상접속망의 위상수학(topology)과 같은 특정한 구성방식특징과 독립이여야 한다.

거친 립도 병렬응용프로그람들은 coarse−grain병렬성을 개발하여야 한다. 그림 4−3에서 례증하는바와 같이 병렬콤퓨터의 통신부가처리는 처리속도보다 훨씬 낮은 률로 증가되고 있는데 이 추세는 계속되고 있다. Coarse−grain병렬프로그람은 현재 병렬콤퓨터의 세대에 비해 더 좋은 확대가능성을 가진다. 확대가능한 콤퓨터제작을 자극하는것은 그것들을 요구하는 과학, 사회경제적중요성과 관련되여 있으며 콤퓨터체계의 빠른 발전으로 하여 응용범위는 계속 넓어 지고 있다.

개인용콤퓨터와 인터네트의 효과성으로 하여 콤퓨터자원들은 지금 전문가만이 아니라 보통 사람들에게 접근하고 있다. 아래에서 확대가능한 콤퓨터를 요구하는 몇가지 중요한 응용클라스들을 고찰한다.

기술적응용들 많은 과학기술분야들은 확대가능한 콤퓨터들 특히 고성능콤퓨터들로부터 많은 리득을 얻고 있다. 콤퓨터이전에 과학은 리론적, 실험적방법들에 의해 연구되였다.

콤퓨터들은 제3의 방법을 제공하는데 그것은 계산생물학, 계산물리학, 계산화학과 같은 새로운 과학분야를 창조하였다.

콤퓨터의 중요성은 Kenneth Wiason에 의하여 강조되였는데 그는 초고속콤퓨터는 갈릴레오망원경다음에 발명된 가장 중요한 도구이라고 말하였다.

아래에 대표적인 과학과 공학적응용들을 렬거한다. 그러한 응용들의 일반적인 특징은 그것들이 계산력 특히는 류동소수점용량에 대해 무제한한 요구를 가지고 있다는것이다.

확대가능한 콤퓨터들은 언제나 짧은 시간동안에 제기된 문제를 풀고 대규모문제를

푸는데 리용된다.

- **생물학** 약품설계, 단백질폴딩, 유전정보학, 인체모의, 가상인간 등에 리용된다.
- **화학** 촉매와 효소의 연구, 화학설계모의 등에 리용된다.
- **물리학** 새로운 물질, 반도체모의, 핵심부융합체계설계연구, 소립자연구 등에 리용된다.
- **천문학** 우주론, 은하계형성, 태양동력학, 흑점충돌, 천문학자료해석 등에 리용된다.
- **기후와 날씨** 날씨예보, 기후예측 등에 리용된다.
- **환경** 오염흐름모형화, 공기와 물질모형화, 생태계, 지구관측체계 등에 리용된다.
- **지구물리학** 지진자료해석, 저수지모형화, 지진예측 등에 리용된다.
- **공학** 운반기구, 건물, 다리의 설계, 소편모의, 기관설계, 고주파회로모의 등에 리용된다.
- **화상과 신호처리** 시공간적응처리, 합성벌린면레이다, 화상실감묘사와 해석, 가상현실 등에 리용된다.

업무응용프로그람 과학/공학계산응용프로그람은 콤퓨터응용시장에서 작은 부분만을 차지하지만 훨씬 큰 부분이 상업적응용프로그람으로 이루어 진다.

그러한 응용프로그람들은 과학/공학응용과 같이 높은 계산력을 요구할수 없지만 종종 큰 기억기와 디스크용량, 높은 I/O들을 요구한다.

또한 높은 RAS(믿음성, 유용성, 장치가능성)를 가진 콤퓨터체계를 요구한다. 전형적인 상업적응용프로그람은 다음과 같다.

- 자료기지관리와 질문
- 직결단위업무처리
- 자료창고화
- 결심채택지원체계
- 제조업과 소매에서 자료발굴

망리용 최근 리용추세는 망중심계산이며 망을 통해 접속된 많은 콤퓨터들에서 응용프로그람이 동작한다. 망은 작은 국부망 혹은 인터네트일수 있다.

주요한 요구는 유효통신, 균형화, 호상운영성, 보안을 포함한다.

이 령역에서 대표적인 응용프로그람은 다음과 같다.

- WWW봉사
- 다매체처리
- 요청비데오
- 전자상거래
- 특정령역에 대한 수자서고

- 원격학습과 공동연구
- 원격의학과 건강관리

처리량 특별히 개발된 병렬응용프로그람은 병렬콤퓨터에서만 동작한다는것은 그릇된 견해이다.

많은 사용자들은 병렬콤퓨터를 대용량기억기와 강력한 처리기를 가진 거대한 워크스테이션으로 보는것이 현실이다. 병렬콤퓨터는 각이한 사용자들의 다중순차프로그람들을 동시에 실행시킨다.

이것은 처리량으로 알려 져 있다.

단일주소(실례로 PVPS, SMPs, DSMs)를 가진 콤퓨터체계들은 다중주소공간기계들(실례로 클라스터와 통보문넘기기 MPPs)보다 우세하다.

응용특징 수천개의 순차응용쏘프트웨어묶음을 별개로 하면서 성숙되고 이식성 있는 유효한 병렬응용들이 출현하였다. 이것은 병렬기계들을 프로그람화하는것이 힘들다는 사실에 기인된다.

다른 리유는 특정응용에 대한 병렬쏘프트웨어의 부족점에 있다. 이것은 그림 4-5에 있는 4차원응용공간에 의해 설명된다.

	정규	비정규		정규	
높은 호상작용 부가처리	1	2	3	4	큰 grain
	5	6	7	8	작은 grain
낮은 호상작용 부가처리	9	10	11	12	
	13	14	15	16	큰 grain
	낮은 병렬성		높은 병렬성		

그림 4-5. 프로그람특징에 의해 분할된 응용령역

응용공간은 알고리듬의 정규성, 병렬성정도, 계산론적정규성, 호상작용부가처리에 따라 다시 분할한다.

이 4개의 특징들은 전체적으로 독립이 아니다. 매개 정방형은 한개의 특별한 응용령역을 표현한다.

오늘날의 대부분의 병렬콤퓨터들은 16개 정방형에 속하는 응용프로그람들을 지원하도록 유리하게 되여 있다. 그 응용프로그람들은 병렬성의 고수준, 거친 립도, 낮은 호상작용부가처리, 정규알고리듬구조를 가진다.

여러가지 콤퓨터들은 각이한 응용령역에서 각이하게 수행할수 있다. 실례로 정방형 11 혹은 12를 탐색하려고 하면 5개의 grain병렬성은 어떤 응용프로그람들에 내장된 대용량병렬성을 리용할수 있다.

일반적으로 말해서 낮은 통신부가처리가 있는 높고 정규적인 구조화된 병렬성을 가진 응용프로그람들은 립도에 관계없이 아주 믿음직하다.

특별한 응용령역을 탐색하는것은 아주 도전적인 과제이다.

그것은 콤퓨터구성방식과 그것의 프로그람작성환경을 개선할것을 요구할수 있다.

4. 2. 처리기설계원리

이 절에서는 고성능처리기설계원리를 고찰한다. 먼저 몇가지 기본처리기용어들을 소개한 다음 처리기구성방식 다시 말해서 CISC, RISC, 초스칼라, 초관흐름, 분리형 CISC/RISC, VLIW와 다매체확장을 고찰한다.

상품화된 극소형처리기계렬들은 4.3절에서 그것들의 성능특징과 비교하여 고찰한다.

Pentium Pro, Alpha 21164는 4.4절에서 실례로서 제시한다.

4. 2. 1. 명령관흐름의 기초

전형적인 명령실행은 본질적으로 4개 단계로 이루어 진다. 즉 꺼내기, 복호화, 실행, 후 쓰기(write−back)들이다.

이 실행단계들은 흔히 명령관흐름이라고 부르는 여러 단계를 가지는 관흐름화된 처리기에 의해 수행된다.

관흐름은 공업적조립흐름선과 같이 많은 순차명령들을 실행한다. 순차명령들은 프로그람순서에 따라 관흐름으로 들어 간다.

비순차실행은 분기 혹은 새치기들을 우연히 만날 때 일어 날것이다.

관흐름화의 기본방법은 시간을 절약하고 통과능력을 증가시키기 위해 련속명령실행을 중복시키는것이다.

명령관흐름설계, 연산(동작), 평가에 관한 기본정의들을 아래에 제시한다.

(1) **관흐름주기 혹은 처리기주기** 관흐름동작을 구동하는 박자주기는 흔히 관흐름에서 단일단계동작을 완성하는 최대시간지연과 같다.

(2) **명령발행(issue)지연시간** 프로그람렬에서 두개의 린접한 명령의 발행사이의 처리기주기수이다.

(3) **CPI(명령당 주기)** 관흐름을 통하여 주어 진 명령을 실행하는데 요구되는 주기수이다. 이 량은 명령의존성이며 따라서 평균CPI는 흔히 프로그람결합에 쓰인다.

(4) **명령발행률** 주기당 발행되는 명령수이다. 이것을 초스칼라처리기의 다중성이라고 부른다.

(5) **단순동작** 단순한 동작들은 실행하는데 한개 주기만을 요구한다. 대부분의 RISC 명령들은 옹근수더하기, 이동, 분기 등과 같이 단순하다.

(6) **복합동작** 복합명령은 실행하는데 많은 주기들을 요구한다. 실례로는 나누기, 대수, 기억기참조 등을 들수 있다.

(7) **자원충돌** 이것은 두개 혹은 그이상의 명령들이 동시에 같은 관흐름의 리용을 요구하는 경우를 가리킨다.

기본관흐름화개념

단일발행처리기는 그림 4-6에 있는바와 같이 4개 단계로 이루어 진 기준선관흐름으로 정의한다. 한개 명령은 한개 주기에서 발행하며 린접하는 발행사이에 한개 주기를 가진다. 그림은 명령에서 순차실행의 리상적인 경우를 보여 준다. 순차실행은 관흐름에서 분기와 새치기의 효과가 생략된것을 가리킨다. 관이 일단 채워 지면 매 명령은 관흐름에서 효과적으로 벗어 나는데 한개 주기만을 요구한다.

분기 혹은 새치기가 프로그람렬에서 생길 때 목표위치에서 다음명령을 꺼내기 위하여 관흐름을 비게 하여야 한다. 이 비순차실행은 매 분기 혹은 새치기로 보상되는 큰 벌칙으로 이끌어 갈수 있다.

분기예측은 흔히 순차벌칙을 감소 혹은 제거시키기 위해 실행된다.

그림 4-6. 순차관흐름화된 명령실행

주기란 무엇인가? 때때로 작업부하는 처리기박자주기수에 의해 표현된다. 고수준언어프로그람의 실행가능한 부분은 명령문렬에 포함된다. 매 명령문은 2진코드로 된 명령렬로 옮긴다. 명령은 기계주기 혹은 관흐름주기라고 부르는 기본단계로서 처리기에 의해 실행된다.

명령에서 실행된 이 기계주기렬을 명령주기라고 부른다.

매 기계주기는 하나 혹은 그이상의 처리기박자주기(간단히 박자로 알려 진)들로 이루어 지는데 처리기박자속도의 역수이다. 극소형처리기에서 처리기내부에 활동체만을 포함하는 기계주기를 내부주기라고 부른다. 처리기모선이 활성되는데 필요할 때 처리기외부로 가는데 필요한 기계주기를 모선주기라고 부른다.

처리기는 수백(혹은 그이상)개의 여러가지 명령주기들을 가진다. 이것은 각이한 명령들이 각이한 명령주기들을 가지며 많은 처리기들이 100개이상의 명령들을 가지기 때문이다.

더우기 같은 명령은 여러가지 CPU상태에서 실행될 때 각이한 명령주기를 가진다.

수백개의 여러가지 명령주기들을 직접 효률적으로 실행하는것은 힘들다. 처리기설계에서 기본기술은 기계주기의 작은 모임을 정의하는것이며 그것은 모든 명령주기들을 실현하는데 리용될수 있다. 매 기계주기는 보통 고정된 박자주기수와 매 박자주기에서 나타나는 조종신호들의 고정모임으로 이루어 진다.

모선주기 대 처리기주기 콤퓨터체계에서 여러가지 부분체계들은 흔히 각이한 박자속도에서 구동된다. 실례로 IBM PC는 250MHz박자속도와 체계모선에 대해서는 66MHz를 가진 처리기를 리용할수 있다. 처리기박자주기는 1000/250=4ns이며 모선박자주기는

15ns이다.

실례 4.2. 명령관흐름과 Pentium의 주기시간

Intel Pentium처리기에서 옹근수명령들은 그림 4-7에서 보여 주는바와 같이 5단계로 실행된다. 매 단계는 기계주기를 실행한다. 리상적인 환경에서 매 관흐름단계는 한개 박자지연만을 가져 온다. 따라서 옹근수명령주기는 5개 관흐름박자만큼 짧아 질수 있다.

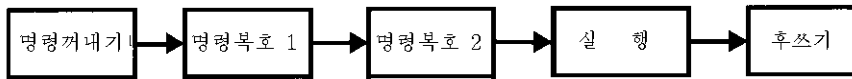



그림 4-7. Intel Pentium처리기에서 명령관흐름

다음명령이 캐쉬에 있으면 명령꺼내기단계는 캐쉬로부터 명령을 꺼내는 내부기계주기를 실행한다.

그러한 기계주기는 한개 박자만을 필요로 한다. 그러나 명령이 캐쉬에 있지 않으면 명령꺼내기단계는 기억기로부터 캐쉬행을 적재하는 모선주기를 실행할것이다.

이 캐쉬채우기모선주기는 5 혹은 그이상의 모선박자를 가지는데 기억기가 얼마나 빠른가에 달려 있다. 수 5는 다음과 같은 사실에서 나온다. Pentium에서 캐쉬준위에 32byte가 있고 자료모선은 64bit(혹은 8bit)폭을 가진다. 리상적인 경우에 캐쉬채우기모선주기는 주소와 다른 신호를 기억기에 보내는데 한개 모선박자, 캐쉬행에서 4×8=32byte를 전송하는데 4개의 추가적인 모선박자를 소비한다.

실례 4.3. Pentium처리기에서 명령실행

다음과 같은 명령이 pentium처리기에서 실행되여 가상기억위치 M에서 자료단어를 작업등록기 EAX로 적재한다고 가정하자.

즉 MOV EAX, M

명령 그자체가 I-캐쉬에 있지만 적재된 자료단어는 D-캐쉬에 있지 않다고 하자. 그러면 명령주기는 그림 4-7에서 보여 준 관흐름을 통하여 다음과 같은 5개 주기들의 렬로 이루어 진다.

(1) I-캐쉬에서 명령꺼내기
(2) 명령복호단계 1실행
(3) 명령복호단계 2와 주소변환실행
(4) D-캐쉬에서 한개 캐쉬행꺼내기
(5) 자료단어를 등록기 EAX에 적재

명령실행순서화 명령들이 언제나 프로그람순서에 따라 실행될 때 그것을 순서실행이라고 부른다.

초기의 처리기들은 모두 프로그람순서대로 명령을 실행하도록 설계되였다. 이 순차순서화는 명령이 자료 혹은 다른 명령을 가진 조종의존성을 계속할수 없을 때 관흐름을 멎게 한다. 순서실행에서 멎기문제가 나타나면 흔히 처리기효률성은 아주 나빠진다.

콤파일러기술들 혹은 전문재순서화기구는 불필요한 관흐름멎기를 제거하기 위해 프로그람렬을 재순서화하는데 쓰일수 있다. 그러한 프로그람실행형식을 비순서실행이라고 부른다.

이것은 분기예측과 주기실행과 함께 동적프로그람실행을 가능하게 한다. 프로그람재순서화는 또한 불필요한 기억기대기를 제거할수 있다.

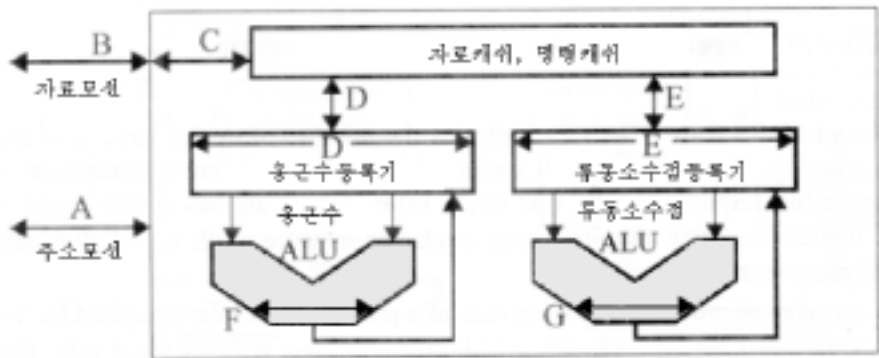



그림 4-8. 처리기설계에서 경로와 단어길이

관흐름설계파라메터들 처리기로서 16b, 32b 혹은 64b처리기를 고찰한다.

이 의미를 그림 4-8에서 준다. 여기서 A는 주소모선의 폭, B는 자료모선의 폭, C는 기억기와 캐쉬사이 자료경로폭, D와 E는 캐쉬와 처리기(등록기들)사이 자료경로폭, D와 E는 또한 각각 옹근수등록기와 류동소수점등록기사이 등록기폭, F와 G는 각각 옹근수 ALUs와 류동소수점 ALUs의 실행폭들이다.

A와 B의 크기는 처리기에 붙은 체계폭은 물론 처리기소편의 핀(pin)계산에 의해 얻어 진다. 파라메터 C, D, E는 관흐름화된 자료경로폭을 결정한다. 파라메터 F와 G는 실행단의 폭을 결정한다. 흔히 모든 이 파라메터들의 크기들은 박자속도와 Mi/s률을 최대화하기 위해 공동으로 결정된다.

처리기와 협동처리기 콤퓨터의 중앙처리기를 중앙처리단(CPU)이라고 부른다. 여기서는 단일 VLSI소편우의 CPUs에만 관심이 있다. CPU는 본질적으로 명령처리기인데 등록기파일캐쉬, 산수 및 론리단(ALU), 류동소수점단, 처리기조종단, 체계모선대면부 등과 같은 다중기능단들로 이루어 진다.

하나 또는 그이상의 협동처리기를 그림 4-9에서 보여 준바와 같이 CPU에 붙인다. 협동처리기는 CPU로부터 발송된 특정한 명령들을 실행한다.

협동처리기는 류동소수점가속기, 벡토르연산수를 실행하는 벡토르처리기, 수자신호처리기(DSP) 혹은 다매체정보를 실행하는 매체협동처리기가 될수 있다.

협동처리기는 혼자서 쓰일수 없다. 그것들은 정규명령들을 조종하거나 I/O동작을 수행할수 없으며 CPU(처리기)에 의해 수행된다.

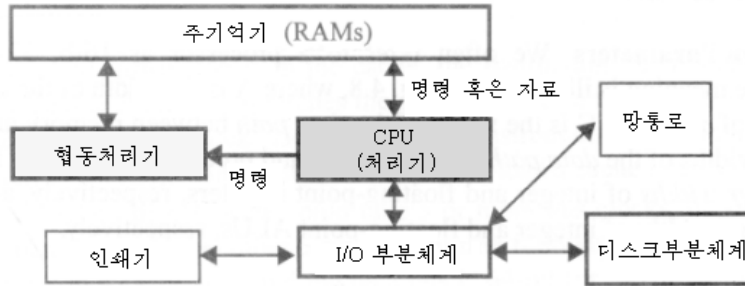



그림 4-9. 처리기와 협동처리기관계

CPI와 MPIS률 CPI는 단일명령을 실행하는데 필요되는 주기수를 가리킨다. 그러므로 CPI는 각이한 명령형태에 따라 변한다. 단순한 동작은 대체로 실행하는데 한개 주기를 요구하며 따라서 1과 같은 CPI를 가진다. 복합명령은 흔히 실행하는데 더 많은 주기들을 요구하며 20개 처리기주기를 쉽게 초과할수 있다.

속도 혹은 처리기명령실행률은 흔히 MIPS률(초당 백만개명령)에 의해 측정되며 박자속도에 따라 선형으로 증가한다.

MIPS률은 주어 진 처리기에서 프로그람을 실행하는데 필요되는 평균CPI에 따라 직접 줄어 든다. 박자속도 fMHz와 평균 CPI를 가진 처리기를 고찰하자. 명령실행률은

$$명령실행률=f\ /\ CPI \qquad (MPIS\ 혹은\ Mi/s에서) \qquad (4.1)$$

로 계산된다.

f=120MHz, 평균 CPI.6이라고 하자.

그러면 처리기속도는 120/0.6=200MIPS이다.

CPI값은 모든 명령형태에 관해 평균화되여야 하며 주어 진 문제통합에서 각이한 명령형태의 빈도에 따라 무게를 주어야 한다. 유럽에서는 MIPS를 Mi/s표기법으로 표현한다. 이 책에서 이 두 표기법들을 서로 엇바꾸어 쓴다.

4. 2. 2. CISC로부터 RISC와 그이상에로

지난 20년동안에 CISC(복합명령모임계산)으로부터 RISC(축소명령모임계산)에로의 이행은 ISA에서 본질적인 변화를 가져 왔다. 아래에서 두 구성방식의 차이점과 류사점들을 고찰하자.

콤퓨터의 ISA는 하드웨어에 의해 직접 실행된 기초명령을 특징 짓는다. 다른 말로 ISA는 주어 진 처리기형태에 의해 실행가능한 기계명령들을 말한다.

ISA의 복잡성은 명령형식, 자료형식, 주소화방식, 일반목적등록기들, Opcode명세서, 리용된 흐름조종기구에 원인이 있다.

이 특징들에서 여러가지 선택에 기초하여 CISC와 RISC는 ISA에 대한 두개의 학문파를 대표하고 있다.

현재 주요회사들에 의해 50여개의 각이한 구성방식들이 발명되였으며 그 구성방식들에 의하여 수백개의 완성품들이 제작되였다.

구성방식차이 RISC와 CISC처리기들의 기본하드웨어구성방식을 그림 4-10에서 비교하였다. 이 도표는 pure RISC와 classic CISC처리기들사이의 구성방식차이를 보여준다.

초기 CISC구성방식은 명령과 자료 두가지를 보유하기 위해 단일화된 캐쉬를 리용하였다. 그것들은 같은 자료/명령경로를 공유하여야 하며 작은 등록기파일을 리용해야 한다. M68030와 68040과 류사하게 최근의 CISC는 자료와 명령에 대해 분할캐쉬를 리용한다.

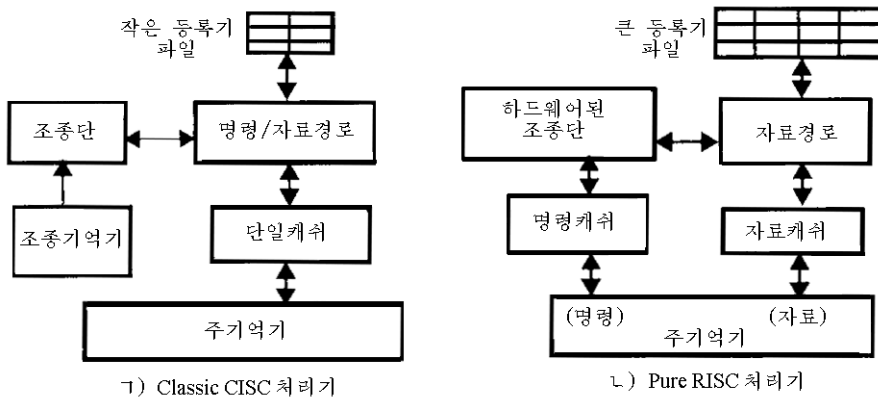



그림 4-10. RISC와 CISC처리기에서 구성방식차이

극소형프로그람조종은 CISC처리기에서 큰 명령모임을 수행하는데 쓰이였다. 그러나 최근의 처리기들은 모두 하드웨어된 조종으로 넘어 갔다.

RISC처리기에서 분할명령캐쉬와 자료캐쉬는 여러가지 접근경로를 리용하며 흔히 대용량등록기파일이 쓰인다. RISC는 소편내장실행을 위해 하드웨어된 조종을 리용하여야 한다. 이것은 관흐름이 과도한 분기와 새치기가 없이 원활히 동작한다면 CPI를 효과적으로 한주기 줄인다.

CISC 도로지도 초기콤퓨터들은 보다 단순하고 작은 명령모임부터 시작하였다. 그러나 명령모임의 복합성은 해마다 뚜렷하게 증가하였다. 이것은 지난 30년동안에 하드웨어비용의 뚜렷한 감소와 쏘프트웨어비용의 증가로부터 나왔다. 망효과는 아주 많은 기능들이 하드웨어지원에 의해 제공된것이였으며 복합명령수가 증가하는데 따라 명령모임을 훨씬 크게 하였다.

CISC개념은 x86계렬, Motorola M6800, Digital VAX계렬 그리고 IBM주콤퓨터에서 잘 표현된다. 명령모임의 증가는 또한 1960년과 1970년에 마크로프로그람화된 조종의 대중성에 의해 촉진되였다. 전형적인 CISC명령모임은 300개 명령을 쉽게 초과할수 있다.

이것은 8b에서 16b, 32b, 64b의 단어길이에 따라 변수명령/자료형식의 리용에 기인

한다. 다만 8~24b의 일반목적등록기의 작은 모임만이 쓰인다.

CISC는 첨수화와 간접주소화를 포함하는 12개이상의 주소화방식에 기초한 많은 기억기참조연산들을 허용한다. 거의 모든 콤퓨터제작자들은 1960년부터 RISC소편이 출현할 때까지(1980년) CISC구성방식에 많이 투자하였다.

RISC도전 CISC구성방식에서 30년의 장성후 콤퓨터사용자들은 ISA와 쓸모 있는 하드웨어/쏘프트웨어기술공학사이 성능관계를 평가하기 시작하였다. 년간 프로그람추적을 통하여 콤퓨터과학자들은 결국 25%의 복합명령모임만이 약 95%의 시간에 자주 쓰인다는것을 발견하였다. 이것은 약 75%의 하드웨어지원명령들은 좀처럼 쓰이지 않는다는것을 의미한다. RISC의 기본방법은 일반실례를 빠르게 하여 이전의 CISC설계에 비해 성능에서 극적인 증가를 얻게 하는것이다.

그러면 자연적인 질문이 나온다. 즉 널리 쓰지 않는 명령에 대해 왜 값비싼 소편을 랑비해야 하는가?

대답은 프로그람혼합에서 낮은 리용빈도를 고려한후 복합명령을 하드웨어보다 쏘프트웨어에 밀어야 한다는것이다.

잘 쓰이지 않는 명령들을 쏘프트웨어에 미는것은 빈 소평령역을 얻어 내여 단일 VLSI소편우에서 전체 처리기를 가능한껏 제작하게 한다. 지어 매 소편내장분할들과 류동소수점단들은 RISC처리기소편에서 제작한다.

RISC명령모임은 대표적으로 고정 32b 혹은 64b명령형식을 가진 100개이하의 명령을 포함한다.

다만 3개~ 5개의 단순주소화방식을 쓴다.

이것의 명령들은 등록기에 기초하는데 등록기로부터 연산수들을 꺼내고 결과자료를 등록기에 후쓰기해야 한다는것을 의미한다.

다만 적재/기억명령들은 캐쉬 혹은 계층기억기로부터 자료에 접근한다.

연산수들은 그것들이 쓰이기전에 작업등록기에 적재되여야 한다. 이 리유로 하여 RISC를 적재/기억구성방식이라고도 불렀다.

RISC처리기는 흔히 두개의 분리된 등록기파일 즉 32옹근수등록기와 32류동소수점등록기를 리용한다.

어떤 RISC처리기들은 100개이상의 등록기를 리용한다.

큰 등록기파일을 리용하는것외에 소편내장분할캐쉬리용은 또한 접근시간을 뚜렷히 짧게 한다.

RISC에서 기본방법은 등록기, 재순서완충기 혹은 자료-캐쉬로부터 연산수들을 직접 꺼냄으로써 대부분의 명령들을 한주기동안에 실행하는것이다.

표 4-2에서 pure RISC와 classic CISC처리기의 기본특징을 요약한다.

일부 구성방식차이는 앞으로 두개의 구성방식들이 기본적으로 같은 기술공학에 따라 변화하기때문에 없어 질것이다.

실례로 분할캐쉬와 하드웨어된 조종은 역시 MC 68060과 Intel Pentium에서 쓰이고 있다.

표 4-2 classic CISC와 pure RISC응용의 특징

특징	Classic CISC Architecture	Pure RISC Architecture
명령형식	변수형식: 16b 32, 64	고정32bit명령
박자속도*	100-226MHz	180-500MHz
등록기파일	8-24일반목적등록기 (GPR)	32-192GPRs옹근수와 류동소수점파일
지령모임크기와 형태	300근방, 48개이상의 명령형식	100근방, 적재/기억을 제외 하고 등록기에 기초함
주소화방식	12개근방, 간접/첨수주소화 방식포함	3~5로 제한, 읽기/쓰기 기억기주소화
캐쉬설계	단일캐쉬를 쓴 초기모형, 분할캐쉬리용	대부분 분할자료캐쉬와 명령캐쉬를 리용
CPI 평균 CPI	1-20주기, 평균 4주기	단순동작에 대해 1주기 평균 1.5주기근방
CPU조종	대부분 마이크로코드작성, 하드웨어구성된 조종리용	조종기억기가 없이 대부분 하드웨어구성된 조종
대표적인 상업화 된 처리기	Inter x86, VAX8600. IBM390, MC68040, Intel Pentium AMD 496, and Cyrix 686	Sun UltraSparc, MIPS R1000, RowerPC 604, HP PA-8000, Digital 21164
* 1997년 최대박자속도에 기초		

ISA에서 20년간의 론쟁

큰 등록기파일, 자료완충기, 분할 I-캐쉬와 D-캐쉬를 리용하는것은 내부자료를 발송시키는데 리익을 주고 중간결과들의 불필요한 기억을 제거한다.

명령모임의 복잡성을 뚜렷하게 줄이는데 따라 RISC처리기를 단일 VLSI소편우에 쉽게 제작할수 있다.

결과로 나타난 리득은 높은 박자속도, 낮은 평균CPI, 낮은 캐쉬비적중벌칙을 포함하며 콤파일러최량화를 위한 더 좋은 기회를 준다.

그러나 CISC에서 RISC에로의 이행은 구성방식에서의 근본적인 변화였다. 주요한것은 일반적인 CISC응용코드에 대해 2진호환성을 없앤것이다.

프로그람추적에 기초하여 CISC프로그람을 등가인 RISC프로그람으로 전환하는것은 코드길이(명령계수)에서 약 40%의 증가를 가져 온다. RISC코드길이에서 증가는 박자속도에서의 증가와 RISC처리기의 평균 CPI에서의 감소보다 훨씬 작다.

RISC와 CISC사이의 론쟁이 끝났다고 볼수 있다. 두 측들은 서로 많은것을 배웠다고 볼수 있다. 지금 RISC와 CISC구성방식사이 경계는 오히려 희미하게 되였다.

적지 않는 처리기들은 지금 같은 회로기술을 가지고 제작한 RISC와 CISC특징 두가지를 가진 혼성방식으로 제작되고 있다.

4. 2. 3. 구성방식개선방법

극소형처리기들은 1971년 Intel 4004, 4–B CPU부터 시작하였다.

이것은 결국 x86극소형처리기계렬로 전환되여 가고 있다.

여러가지 처리기계렬들을 그림 4-11에 있는 CPI에 대한 박자속도의 설계공간에서 고찰하자.

처리기기술이 빨리 발전하는데 따라 박자속도는 1980년의 10MHz로부터 1997년의 300MHz로 증가하였다.

오른쪽 아래구석에 있는 그늘진 령역은 앞으로의 처리기설계와 아주 비슷하다.

단일발행 CISC 혹은 RISC처리기들은 설계공간의 왼쪽우에 있다. 평균 CPI값을 CISC처리기에서의 10주기로부터 RISC기계에서의 1주기이하로 줄이는것이 요구된다. 평균CPI는 개선된 처리기구성방식과 콤파일러최량화에 의해 낮아 질수 있다.

총체적인 설계목적은 CPI를 작은 주기로 낮추고 동시에 박자속도를 증가시키는것이다.

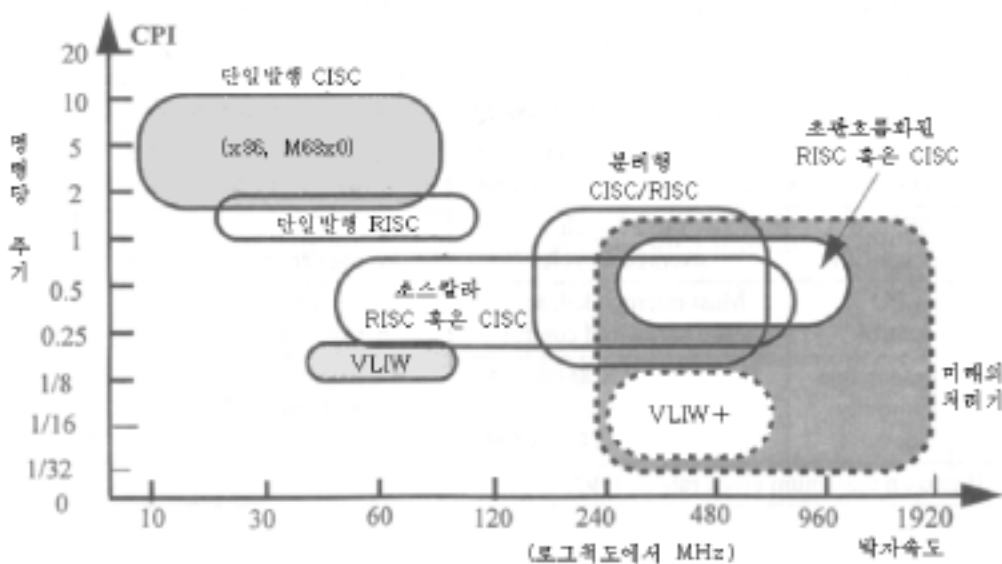



그림 4-11. 1980년부터 2000년까지의 상품극소형처리기의 설계공간

초관흐름화된 설계와 초스칼라 오늘 모든 단일 발행처리기들은 다중발행초스칼라구성방식을 가정하는데로 나가고 있다.

실례로 CPU소편들은 두개의 발행초스칼라 CISC구성방식을 가정하는 M68060와 Pentium을 포함하며 Power PC601과 Pentium Pro는 3개의 발행 RISC처리기를 가정하고 있다 .현재 초스칼라처리기들은 Alpha 21164, Power PC604e, HP PA-8000, MIPSR 10000, Ultra Sparc-2처리기들을 포함하여 주기당 대체로 4개의 명령을 발행한다.

초관흐름처리기들은 아주 높은 박자속도로 구동되는 깊은 관흐름을 리용하는것들이다. 초관흐름화는 Cray초고속콤퓨터로 개발된 다중위상박자화기구를 써서 실현된다. 깊은

관흐름은 많은 단계를 가지는데 초기 처리기관흐름설계의 몇개 단계들로부터 다시 갈라진다.

초스칼라와 초관흐름화설계개념을 적용한 일부 처리기들은 CISC을 가정하든가 RISC를 가정하든가 혹은 혼성구성방식을 가정한다. 이것은 그림 4-11의 중간에 있다.

CISC와 RISC의 분리 이 방법의 기본우점은 x86사용자들에 대하여 보다 좋은 쏘프트웨어호환성을 가진다는것이다.

분리된 구성방식은 그림 4-11의 오른쪽 중심의 겹쳐 지는 면적에서 보여 주는바와 같이 CISC, RISC, 초관흐름화, 한개 미소소편에서의 초스칼라설계의 우점을 결합한다. 이 구성방식은 pure RISC 혹은 classic CISC를 가정하지 않지만 혼성CISC/RISC구성방식을 가정한다.

가장 좋은 실례는 4.4.2절에서 고찰된 Pentium Pro핵심부이다.

그 기본방법은 x86코드를 초스칼라와 혹은 초관흐름화된 실행을 수행하는 뒤단(backend)RISC핵심부를 위한 RISC형명령으로 옮기는 CPU소편에서 앞단(frontend)부분을 가지는것이다. 이것을 분리형 혹은 혼성 CISC/RISC구성방식이라고 부른다.

VLIW/콤파일러방법 이 처리기구성방식은 두개의 잘 째인 개념 즉 수평마이크로프로그람작성과 초스칼라실행개념으로부터 일반화된다. 전형적인 VLIW(최장명령단어, very long instruction word)명령은 수백 b(다중흐름콤퓨터설계에서 256 혹은 1024b와 같은[241])를 가진다.

매 VLIW는 많은 수의 독립동작들로 이루어 지는데 다중기능단들에 의해 병렬로 실행될수 있다. 그것은 그림 4-11의 왼쪽 아래구석에 그려 져 있는데 8-통로(way)병렬성을 가진 다중흐름설계에 대응한다.

VLIW처리기는 짧은(32b) 단어로 씌여 진 사용자코드의 아주 높은 명령준위병렬성(ILP)을 개발할수 있다. 매 VLIW명령에서 모든 동작들은 8-통로, 16-통로 혹은 병렬성의 높은 정도를 나타낸다.

VLIW의 성공은 많은 독립 혹은 비관련 짧은 명령들을 VLIW명령으로 재조직하고 묶는 지능콤파일러에 의거한다.

이 VLIW방법은 초스칼라처리기보다 훨씬 낮은 CPI에 귀착될수 있다. CPI는 32-통로 VLIW구성방식에 대해 1/32로 낮아 질수 있으며 미래의 가능한 처리기설계로 된다. 4.5.3절에서 연구한다.

4. 3. 극소형처리기의 구성방식계렬

PCs와 탁상(desktop)에서 미소소편의 리용이 널리 보급됨에 따라 소편회사들은 극소형처리기에 기초한 하드웨어, 쏘프트웨어와 응용에 대한 연구와 개발에 막대한 자원을 투자할수 있다.

결과 상품화된 극소형처리기들은 IBM주콤퓨터 혹은 Cray초고속콤퓨터들에서 쓰인 주문설계된 처리기들의 성능에 가까와 가고 있다.

4. 3. 1. 주요구성방식계렬

그림 4-12에 일반목적과 매몰형응용을 위한 주요 극소형처리기구성방식들을 요약한다.

오늘날의 일반목적극소형처리기는 단어길이를 32b 혹은 64b로 가정한다. 1997년 가격시세에 기초하면 그것들은 75$와 500$사이의 높은 가격으로 판매되였다.

CISC는 x84, M680x0구성방식으로 표현된다.

x 86은 1978년부터 7세대의 발전을 거치였다. 1995년 x86소편의 전 세계적인 적재(shipment)는 7천만에 도달하였다.

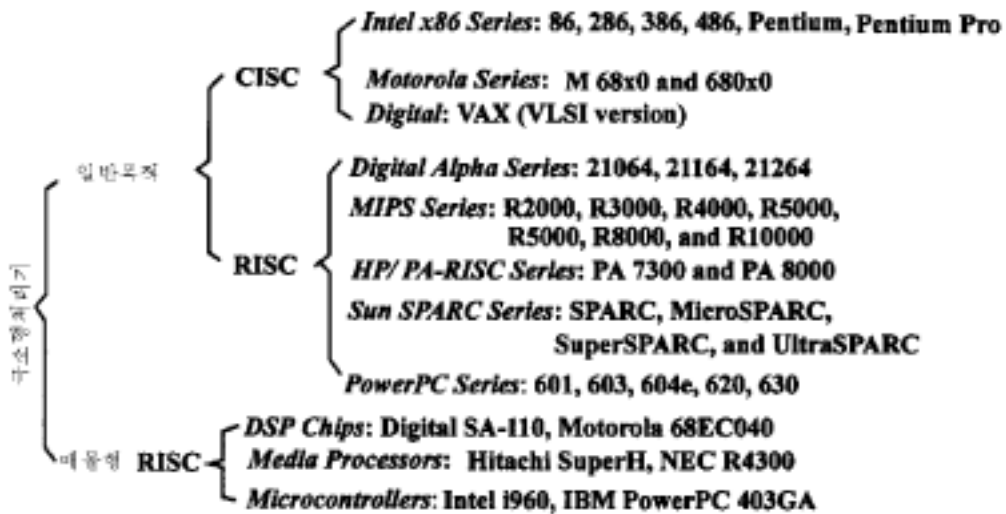



그림 4-12. 극소형처리기계렬과 대표적인 CPU소편

계렬나무로 렬거하면 5개의 주요 RISC구성방식이 있다. 그것을 보면 수자반도체에 의해 발명된 Alpha, MIPS기술에 의한 MIPS, Hewlett Packard에 의한 PA-RISC, Apple, IBM, Motorola련합에 의한 Power PC, Sun Microelectronies에 의한 SPARC 등이다. 매 구성방식을 써서 많은 제작자들은 탁상형, 휴대형의 체계들을 위한 여러가지 도구들을 생산하였다.

고전적인 CISC와 초기의 RISC처리기들은 모두 단일관흐름자료경로를 리용하는 단일발행처리기들이였다.

단일발행 CISC 고전적인 처리기들이란 Intel 86, 286, 386, 486 ; Motorola M680x0계렬; Digital의 VAX/8600 그리고 단일발행 CISC구성방식을 따르는 IBM 370/390과 같은것들이다. 이 처리기들은 모두 1 혹은 2 주기당 단일명령을 발행하도록 설계된다.

그것들은 1970년에 10MHz박자이하로 만들어 졌는데 1980년에는 100MHz이상으로 점차 증가되였다. 486과 M68040은 여기에서 CPU소편의 마지막값을 표현한다.

단일발행처리기의 CPI는 1에서 20주기사이에 있으며 마크로구성방식에 의거한다.

평균값은 명령당 1.5와 10주기사이에 놓이며 ISA와 처리기기술에 의거한다. 이 처리기들은 그림 4-11의 왼쪽 웃구석에서 보여 준다.

단일발행 RISC 그림 4-13에서 RISC구성방식의 4가지 변종을 보여 준다. 즉 단일발행 RISC, 초스칼라 RISC, 관흐름화된 RISC, 분리형CISC/RISC들이다.

RISC처리기들은 Berkeley RISC와 1980년 이후에 개발된 Stanford MIPS의 단일발행설계부터 시작하였으며 그 방법은 Sun SPARC, Micro SPARC MIPS R2000, R3000 등에서 나왔다. 이 처리기들은 1980년에 25MHz의 박자로 만들어 졌으며 1992년에 120MHz로 점차 올라 갔다.

하드웨어된 조종에 의해 대부분 명령들의 CPI는 바로 1주기로 감소되였으며 평균 CPI는 대부분의 응용프로그람들에서 1.5주기아래에 있다. 그러나 단일발행 RISC의 CPI는 박자가 얼마나 빠른가에 관계없이 더 낮아 질수 없다.

CISC이상의 개선은 역시 Intel i860과 Digital 21064에서의 초스칼라 RISC설계가 출현할 때까지 제한되였다.

4. 3. 2. 초관흐름처리기 대 초스칼라처리기

처리기들의 성능을 개선하기 위한 두가지 기본기술들은 더 높은 ILP를 개발하거나 명령관흐름을 보다 간단한 단계들로 세분화하여 박자속도를 증가시키는것이다. 이 두가지 기술들을 그림 4-13에 제시하는데 그것은 서로 지원하게 하며 때때로 보충한다.

발행률증가 초스칼라처리기에서 다중명령들은 주기당 발행되며 주기당 다중관흐름에 의해 다중결과들이 생성된다. 초스칼라처리기들은 사용자프로그람에서 ILP를 개발하기 위해 설계된다. 독립적인 명령들만이 대기상태를 일으킴이 없이 병렬로 실행될수 있다.

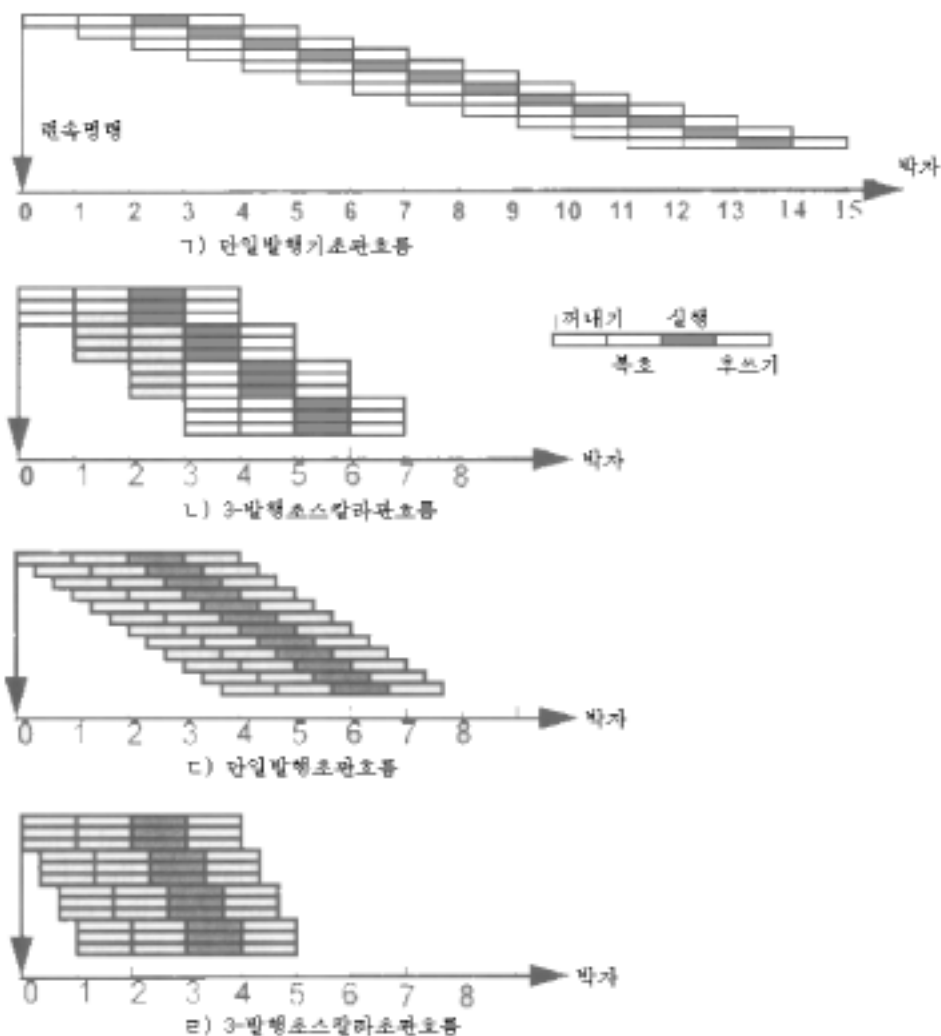
ILP량은 프로그람에서 프로그람으로 변한다. ILP의 평균값은 보통 프로그람추적에서 2와 5사이로 측정되였다. 일부 전문코드들은 과학 Fortran코드에서 500 혹은 그보다 더 높은 ILP를 포함할수 있다. 이 리유로 하여 명령발행률은 현재 초스칼라설계에서 2~6으로 제한되였다.

단일발행 RISC처리기들은 Digital의 21064, M88110, Micro SPARC2, Intel i860, Pentium, PA7200, MIPS R5000, Cyrix 6x86(M1과 M2), M68060 등에서 두 발행처리기들의 도입으로 점차로 바뀌여 졌다.

이 미소소편들은 초스칼라 RISC와 x86초스칼라 CISC처리기들을 포함한다.

관흐름설계선택 그림 4-13에서 명령관흐름설계를 위한 4가지 방식을 보여 준다. 모든 4개의 관흐름설계들은 4가지 단계로 규정되며 기초관흐름박자주기의 배수인 박자속도로 구동된다.

경우 ㄴ), ㄹ)에서 초스칼라설계는 병렬로 동작하는 3개의 기능적인 관흐름리용을 보여 준다.

그림 4-13. 기초관흐름, 초스칼라, 초관흐름처리기들에서 관흐름동작들

경우 ㄷ)와 ㄹ)는 두개의 초관흐름화된 설계에 대응하는데 박자주기는 단지 경우 ㄱ)와 경우 ㄴ)의것의 1/3이다.

이 관흐름동작에서 12개 명령을 실행하는 4개 방식을 보여 준다.

실행시간은 경우 ㄱ)에서 경우 ㄹ)까지는 각각 15주기로부터 7주기, 7.66주기, 5주기로 감소한다.

충분히 큰 작업부하로 경우 ㄴ)는 경우 ㄱ)보다 가능한 3배 더 빠르다.

경우 ㄹ)는 경우 ㄱ)보다 9배 더 빠르거나 경우 ㄴ) 혹은 경우 ㄷ)보다는 3배 더 빠르다.

4개 처리기설계의 처리량은 류사하게 평가될수 있으며 같은 시간제한이 주어 진다.

일반적으로 *m*−발행초스칼라처리기는 같은 박자속도와 같은 관흐름깊이를 가정하면

단일발행처리기보다 적어도 *m*배 더 빠르다. *n*배 더 빠른 박자를 가진 초흐름화된 처리기는 기초관흐름보다 적어도 *n*배 더 빠를수 있다. *n*배 더 빠른 박자[경우 ㄹ)]로 구동된 *m*-발행처리기에서 결합속도는 경우 ㄱ)에 있는 기초관흐름보다 가능한 *mn*이다.

초스칼라 x86구성방식 오늘날의 x86극소형처리기들은 Pentium과 Pentium Pro의 여러가지 버션을 포함하는데 모두 초스칼라처리기들이다. 표 4-3에 요약하여 제시하였다.

Intel, AMD, Cyrix의 3가지 x86계렬들이 이 표에서 비교된다.

거의 모든것들은 단일화된 64-KB캐쉬를 리용하는 CyrixM_2을 제외하고 분할 I/D캐쉬를 리용한다.

복호률은 Pentium과 Cyrix 6x86설계의 전통적인 초스칼라설계에서의 발행률과 같다.

Pentium Pro, Pentium?2, AMD K5/6의 설계에서 복호률은 마이크로명령들이 (uops)기능단들로 발행되기때문에 발행률보다 낮다.

3개의 Pentium모형(P54C, P54CS, P55C)들은 모두 순서실행만을 허용하는 두개 발행 초스칼라설계들이다.

표 4-3 초스칼라극소형처리기의 기본특징들

판매자	Intel						AMD		Cyrix	
모형/특징	Pentium			Pentium Pro		Pentium-2	K5	K6	6x86	
	P54C	P54CS	P55C	96	96S				M1	M2
최대박자 속도(MHz)	120	200	200	150	200	233-266+	100	>180	150	225
Pin-Out	P54C	P54C	P54C	PPro	PPro	Pentium-2	P54C	P55C	P54C	P55C
I/D캐쉬 (KB)	8/8	8/8	16/16	8/8	8/8	N.A.	8/16	32/32	16/16	64Ufd
MMX	아니	아니	예	아니	아니	예	아니	예	아니	예
복호률	2	2	2	3	3	3	1-4	2	2	2
발행률*	2 instr.	2 instr.	2 instr.	5 uops	5 uops	5 uops	4 uops	4 uops	2 instr.	2 instr.
비순서실행	아니	아니	아니	예	예	예	예	예	Ltd	Ltd
소편크기 (mm^2)	148	90	140	308	196	N.A.	181	~180	167	<200
3극소자 (백만개)	3.3	3.3	4.5	5.5	5.5	N.A.	4.0	8.8	3.3	6.0
기술	Bi-CMOS		CMOS	Bi-CMOS		CMOS				
처리 (μm/layer)	0.5/4	0.35/4	0.28/4	0.5/4	0.35/4	0.28/4	0.35/3	0.35/5	0.44/5	0.35/5

* x86명령을 가리킨다. Ufd: Uniied, N. A.(not available),

\+ 256-KB 2차캐쉬를 포함한다. Ltd: Limited, upos: micro-operation

Pentium에서 U와 V관흐름들은 서로 의거한다. 한개 관흐름이 멎을 때 다른 관흐름은 대기하여야 한다. 이것은 초기 Pentium모형에서 주기를 잃어 관흐름을 위험하게 하는 엄중한 결함을 발로시킨다. 최근의 성능평가기준에 기초하여 Pentium은 i486속도를 2~3배 증가시켰다. 두개의 Pentium Pro(96과 96 S)들은 Pentium의 속도를 두배로 하였다. Pentium Pro는 속도성능에서 i486을 4~6배로 증가시켰다.

표 4-4 높은 급 초스칼라극소형처리기의 기본특징

특징	Digital 21164	Power-PC620	Power-PC 304e	Ultra-Sparc-2	Micro Sparc-2	HP PA-8000	MIPS R10000	MIPS R5000	Pentium Pro
최대박자 (MHz)	500	200	225	250	110	180	200	180	200
캐쉬크기 +(KB)	8/8/96	32/32	32/32	16/8	16/8	None	32/32	32/32	8/8
발행률	4	4	4	4	2	4	4	2	3
관흐름관계	7-9-12	5	6	6-9	5	7-9	5-7	5	12-14
비순서실행	6 loads	16 inst.	16 Instr.	None	None	56 instr.	32 instr.	None	40 ROBs
이름교체 등록기	None 8Fp	*Int 8Fp	12Int	None	None	56 total	32Int 32Fp	None	40 total
기억기 대역너비 (MB/s)	~400	1,200	~180	1300	~100	768	539	~160	528
Package, Pim count	CPGA -499	CAGA-625	CBGA-255	PBGA-521	CPGA-321	LGA-1085	CPGA527	SBGA 272	MCM-387
프로쎄스 (μm/ layers)	0.35/4	0.35/4	0.35/4	0.29/5	0.4/3	0.5/4	0.35/4	0.35/3	0.35/4
소편크기 (mm^2)	209	240*	148	149	233	345	298	84	196
3극소자	9.3m	6.9m	5.1m	3.8m	2.3m	3.9m	5.9m	3.6m	5.5m
전력(W)	25	30	20*	30	9	>40	30	10	35**
SPEC95++ Integer/FP	12.6/18.3	9.0/9.0*	8.5/7.0	8.5/15	1.4/1.9	10.8/18.3	8.9/17.2	4.0/3.7	8.7/6.0

\+ 명령캐쉬/2차캐쉬/2차캐쉬 ++ SPEC95integer/SPEC95floating-point

* 마이크로설계자원평가 ** 512-KB 2차캐쉬 ROB: 재순서완충기

Int: 옹근수 Fp: 류동소수점

Pentium−2(Klamath라고도 부른다.)는 MMX(Multimedia Extension)특징을 보충하여 Pentium Pro를 발전시켰다.

AMDK6와 Cyrix 6x86M_1은 성능에서 Pentium Pro와 비교할수 있다. AMDK6와 Cyrix M_2은 K_5와 M_1에서 각각 발전시킨 MMX에 대응한다.

초스칼라 RISC처리기들 대부분의 초스칼라설계들은 CISC대신에 RISC방법을 선택한다. 발행률은 Power PC 601, i960, Pentium Pro, Pentium−2 등에서 3배로 증가하였다.

오늘날의 초스칼라 RISC설계들은 표 4−4에서 보여 주는바와 같이 대부분 4−발행설계로 넘어 갔다.

이 표는 1997년까지 상품화된 극소형처리기에서의 초스칼라설계를 요약한것이다.

초스칼라처리기에서 발행률증가는 평균 CPI률을 발행률 2.3, 4에 대해 각각 최소값 0.5, 0.33, 0.25로 가까와 가도록 낮출수 있다는것을 의미한다.

분리형CISC/RISC CMOS는 미소소편을 제작하는데서 확대가능한 VLSI기술이라는것이 증명되였다.

분리형구성방식은 같은 CPU소편에서 CISC와 RISC 두가지 구성방식으로부터 긍정적인 특징들을 통합한다. 분리형구성방식은 현재 Pentium Pro라고 부르는 Intel P_6의 발전으로 대중화되였다. P_6의 구체적인 연구는 다음절에서 한다.

류사한 분리된 처리기설계는 AMD K_6, Cyrix M_1과 MIPS R1000에서도 출현하였다. 그 특징들을 표 4−3에 제시하였다.

극소형처리기성능 그림 4−14에서 6개의 초스칼라 RISC 혹은 분리형 CISC/RISC극소형처리기의 SPEC95성능을 비교한다.

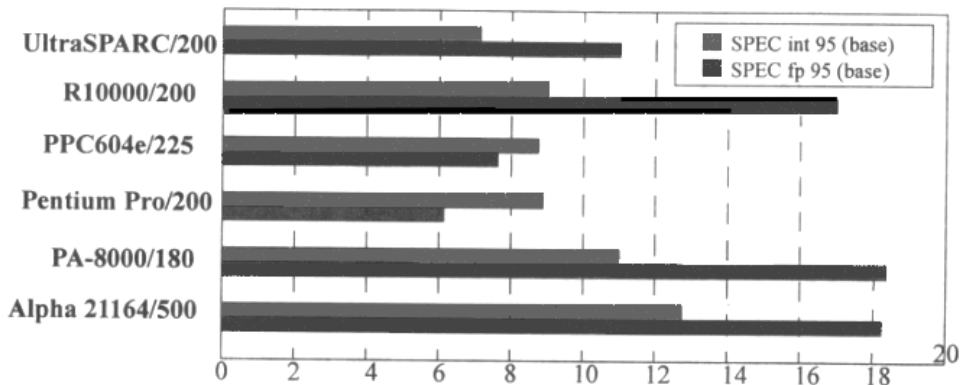



그림 4−14. log척도에서 상품화된 극소형처리기의 SPEC int 95와 fp95(base)성능
(M.slater,IEEE Micro,Dec.1996[566])

매 처리기의 박자속도는 가로축을 잘라서 본다.

Alpha 21164, PA−8000 그리고 R10000은 비교가능한 류동소수점성능을 가진다.

옹근수성능에서 차이는 아주 작다. Power PC와 Pentium Pro의 설계는 주로 보다 좋은 쏘프트웨어호환성을 가진 업무응용프로그람을 위한것이다. 6개 가운데서 Alpha소

편은 성능과 박자속도에서 가장 높은 자리를 차지한다. 그러나 Alpha는 성능/비용률에서 가장 낮은 자리를 차지한다. Alpha 21164, PA-800, R10000는 비교가능한 SPECfp95 성능을 가진다. SPEC int95성능차이는 R10000, Power PC604e, Pentium처리기가운데서 오히려 작다.

4. 3. 3. 매몰형극소형처리기들

매몰형처리기들은 전자공학을 수자화하였다. 제작자들이 그것들을 대량적으로 생산

표 4-5 선택된 매몰형극소형처리기(M.slater, IEEE Micro, Dec.1996[566])의 기본특징

제작자	Digital	VLSI	NEC	Hitachi	IBM	Motorols			Intel	
모형	SA-110	ARM 710	R4300	SH 7604	PPC 403GA	860 DC	68EC 040	CF 5102	960 JA	960 HT
구성방식	ARM	Strong-ARM	MIPS	SuperH	Power-PC	Power-PC	68000	Cold-Fire	I960	I960
박자 (MHz)	200	40	133	20	33	40	40	25	33	60
I/D 캐쉬 (Kbytes)	16/16	8/8	16/8	4/4	2/1	4/4	4/4	2/1	2/1	16/8
FPU	아니	아니	예	아니	아니	아니	예	아니	아니	아니
MMU	예	예	예	아니	아니	예	예	아니	아니	아니
모선주파수 (MHz)	66	40	66	20	33	40	40	25	33	20
MIPS+	230	36	160	20	41	52	44	27	28	1008
전압 (core/bus)	2.0/3.0	5	3.3	3.3	3.3	3.3	5	3.3	3.3	3.3
전력 (mW)	900	424	2200	200	265	900	4500	900	500	45020
MIPS/W	239	85	73	100	155	58	10	30	56	22
3극소자 (백만개)	2.1	0.6	1.7	0.45	0.58	1.8	1.2	N.A.	0.75	2.3
프로쎄스 (μm/layers)	0.35/3	0.6/2	0.35/3	0.8/2	0.5/3	0.5/3	0.65/3	0.6/3	0.8/3	0.6/4
Die 크기 (mm^2)	50	34	45	82	39	25	163	N.A.	64	100

+MIPS(million instructions per second)은 Dhrystoen 2.1에 기초,
FPU: 류동소수점단 MMU: 기억기관리단

하였다. 표 4-5에 선택적으로 렬거된것이 일부 대표적인 매몰형처리기모형과 그것들의 특징들이다.

매몰형처리기들은 주로 DSP(digital signal processing), 매체처리, 극소형조종기응용들을 위한것이다.

이 처리기들은 단어길이가 4b에서 8, 16, 32b의 넓은 범위에 있다. 매몰형제품의 단가는 5$에서 40$사이에서 많이 변한다. 4b와 8b매몰형소편들이 세탁기, 휴대용전화기, 원격조종, 전자완구와 매몰형조종목적을 위한 많은 작은 장치들에 널리 쓰이였다.

실례로 4b처리기소편들의 전 세계적인 판매는 1996년 말까지 16×10억을 초과하였다.

16b와 32b매몰형소편들은 DSP, 인쇄기, 다매체 그리고 조종응용을 위한것들이다. 1995년에 32b Hitachi SuperH적재는 영상유희시장의 막대한 수요로 하여 혼자서 14×100만을 초과하였다.

4. 4. 극소형처리기의 실례연구

우리는 여기에서 지난 10년동안 RISC/CISC에서 진화적변화의 두 극단을 표현하는 두개의 극소형처리기를 연구한다.

Digital의 Alpha 21164처리기는 한 극단에 있는데 일반적인 초스칼라 RISC구성방식에 적합하다. 그것은 RISC명령을 순서대로 실행한다. Alpha설계자들은 박자속도를 가장 빠르게 하였는데 CMOS기술을 제공할수 있다.

다른 극단에는 Pentium Pro처리기가 있는데 4.4.2절에서 서술된바와 같이 분리형 CISC/RISC방법을 재순서화된 마이크로동작을 실행하는데 리용하였다.

Pentium방법은 PC와 노트형콤퓨터공업에서 현재 x86ISA와의 쏘프트웨어호환성을 더 강조한다.

4. 4. 1. Digital의 Alpha 21164극소형처리기

이 4-발행Alpha소편은 Digital의 64b AXP구성방식에 기초하며 성능은 초스칼라 RISC극소형처리기가운데서 제일 높다. 21164의 구성방식은 그림 4-15에서 보여 준다.

4개의 실행단, 곱하기등록기파일, 분할 I/D캐쉬 그리고 0.35um CMOS소편에 2차캐쉬가 있다.

CMOS소편은 외부3차캐쉬와 다중처리기 snoopy모선일관성조종에 접근하기 위한 매몰형캐쉬조종과 모선대면부(CBOX라고 부른다.)를 가진다.

처리기는 6개의 재순서화된 적재동작을 제외하고 순서프로그람들을 실행한다. 그러나 21164는 nonread명령들을 순서가 없이 실행할수 없다. 대부분의 다른 초스칼라처리기는 명령혼합을 순서가 없이 실행할수 있도록 설계한다. 이런 의미에서 AXP구성방식은 ILP를 개발하는데서 오히려 보수적이다.

고속박자속도 Digital은 Alpha계렬에서 최대박자속도방법을 선택하고 1992년에는

200MHz 21064, 1994년에는 275MHz 21064A, 1995년에는 300MHz 21164, 1996년에는 500MHz 21164A부터 시작하였다.

회사는 500MHz21164를 적재하였으며 대부분의 다른 CPU소편들은 아직 표 4-4에 렬거한바와 같이 200~250MHz로 제한되여 있다. 이 CPU소편은 그림 4-14에서 보는바와 같이 가장 높은 옹근수와 류동소수점 SPEC95성능을 주지만 성능박자속도에서 경쟁자와의 간격에는 비례하지 않는다.

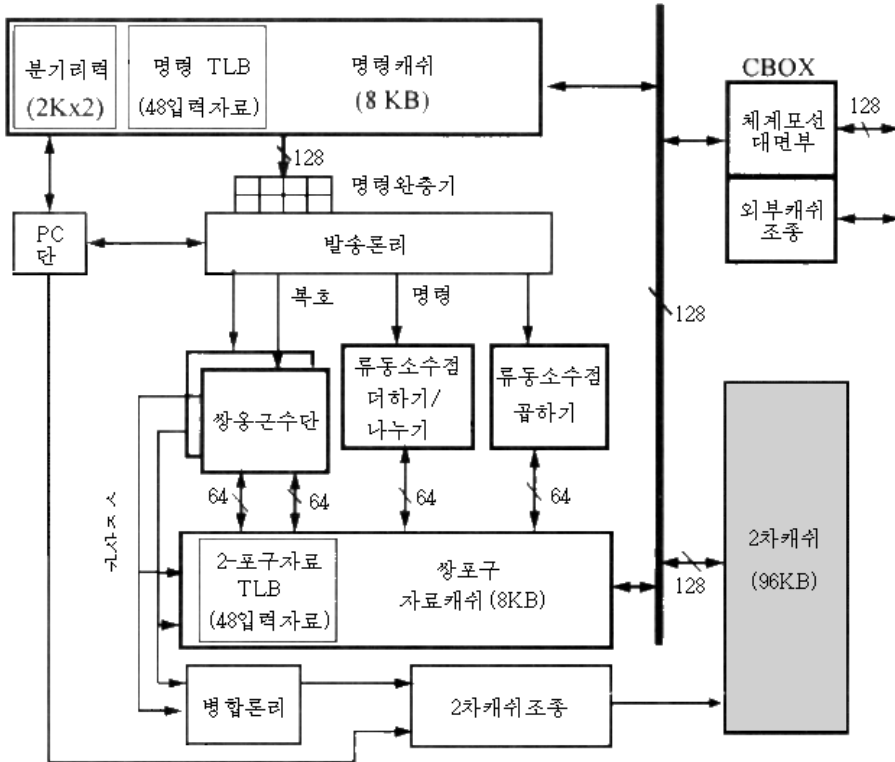



그림 4-15. Digital의 Alpha21164처리기구성방식

미소소편고밀도 RISC소편은 0.35-um CMOS기술에서 밀집도가 가장 높은 10×백만개수의 3극소자를 가지고 있다. 그러나 2×백만이하는 RISC론리부분에 배당되여 있고 나머지 8×백만은 I/D캐쉬와 2차캐쉬를 포함하는 소편내장기억기를 위해 배당되여 있다. 론리적배당은 대부분의 높은 급처리기들보다 낮다.

Pentium Pro는 4.4×백만을 CISC론리에 배당하였으며 그 나머지에 5.5×백만개의 3극소자를 배당하였다.

1997표준에 기초하여 CPUcore는 전형적으로 2~4×백만개의 3극소자를 론리부분에 배당하였으며 1~6백만개의 3극소자를 소편기억기에 배당하였다.

캐쉬계층 캐쉬방책은 Digital이 소편우에 2차캐쉬계층을 가진 극소형처리기만으로 묶어 놓은데서 세운 령역이다. 개개의 8-kbyte의 명령캐쉬와 자료캐쉬는 500MHz박자속도에

서 1주기안에 접근될수 있다.

낮은 96−KB 2차캐쉬는 외부캐쉬보다 고속접근을 제공한다. 매몰형외부캐쉬조종에 따라 대표적인 21164체계는 외부준위 3차캐쉬를 크기에서 1Mb에서 64MB까지 지원한다. 감소된 캐쉬접근시간은 소편성능향상을 일정하게 지원한다.

자료경로폭 옹근수와 류동소수점실행단위는 매개가 64b폭이다. 명령관흐름은 매 단계에서 128b폭이다.

자료를 2차캐쉬에로 보내거나 받는 모선은 128b폭인데 1차캐쉬를 채우는 모선들을 포함한다. 결과 비직소편 3차캐쉬 혹은 기억기에 접근하는것은 Alpha처리기묶음에서 128 자료 pin을 통하여 수행된다.

기능적관흐름 명령관흐름은 7개의 기능단들로 이루어 진다. 즉 명령 F/D단, 두개 옹근수관흐름, 두개 류동소수점관흐름, 기억기주소이행단(MATU)과 그림 4−15에서 본 CBOX이다.

이 관흐름토막들은 다음과 같은 특정기능들을 수행한다.

- 4−단계 명령꺼내기/복호(F/D)단은 꺼내기, 복호화명령, 발행명령 그리고 은페명령을 이룬다. 그것은 BHT(분리리력표, branch history table)와 명령 TLB(translation lookaside buffer)을 포함하는 I−캐쉬를 가지고 동작함으로써 새치기, 제외 그리고 분기예측과제를 조종한다. I−캐쉬는 단계 S0에서 접근되며 옹근수등록기파일(IRF) 혹은 류동소수점등록기파일의 접근은 S3단계에서 수행된다.
- 두개 옹근수실행관흐름은 매개가 3개의 관흐름단계를 가지며 모든 옹근수 RISC 명령들을 실행하고 적재/기억, 분기 그리고 조종명령을 조종한다. 여기서 류동소수점결과에 의해 결정된 조건적분기는 제외한다. 두개 옹근수관은 같은 IRF를 리용한다. 옹근수곱하기는 어느 하나의 옹근수관흐름에 붙인다. 총체적으로 옹근수 RISC명령들은 7개의 관흐름단계에서 실행한다.
- 두개의 4−단계류동소수점관흐름이 있다: 하나는 류동소수점나누기에 붙은 류동소수점더하기관이고 다른 하나는 류동소수점곱하기관이다. 전체적으로 매 류동소수점명령은 명령관흐름을 통해 흐르는 9개의 주기를 가진다.
- 21164는 8−단계기억기접근관흐름을 가지는데 기억기주소이행을 수행하고 적재/기억과 기억기장벽명령을 실행한다. 이 관흐름은 자료캐쉬, 2차캐쉬, MATY 그리고 기억기접근충돌을 조종하고 캐쉬일관성규약을 수행하는 CBOX를 가지고 동작한다.

비차단 캐쉬접근 D−캐쉬에서 자료를 적재하는데 2주기 걸리며 2차캐쉬에서 자료를 접근하는데 4이상의 주기가 걸린다. 단계 S10와 S11에서 2차캐쉬로부터 32B의 캐쉬행을 가진 자료/명령캐쉬를 채우는데는 보충적으로 2주기가 걸린다. 총체적으로 읽기/기억명령은 전체 명령관흐름을 통하여 흐르는데 6~12주기가 걸릴수 있다. 관흐름은 캐쉬비적중(miss)에서 멎지 않는다. 관흐름은 오직 자료의존성이 명령들사이에 존재할 때에만 멎는

다. 매 관흐름주기는 500MHz박자인 경우 2ns이다.

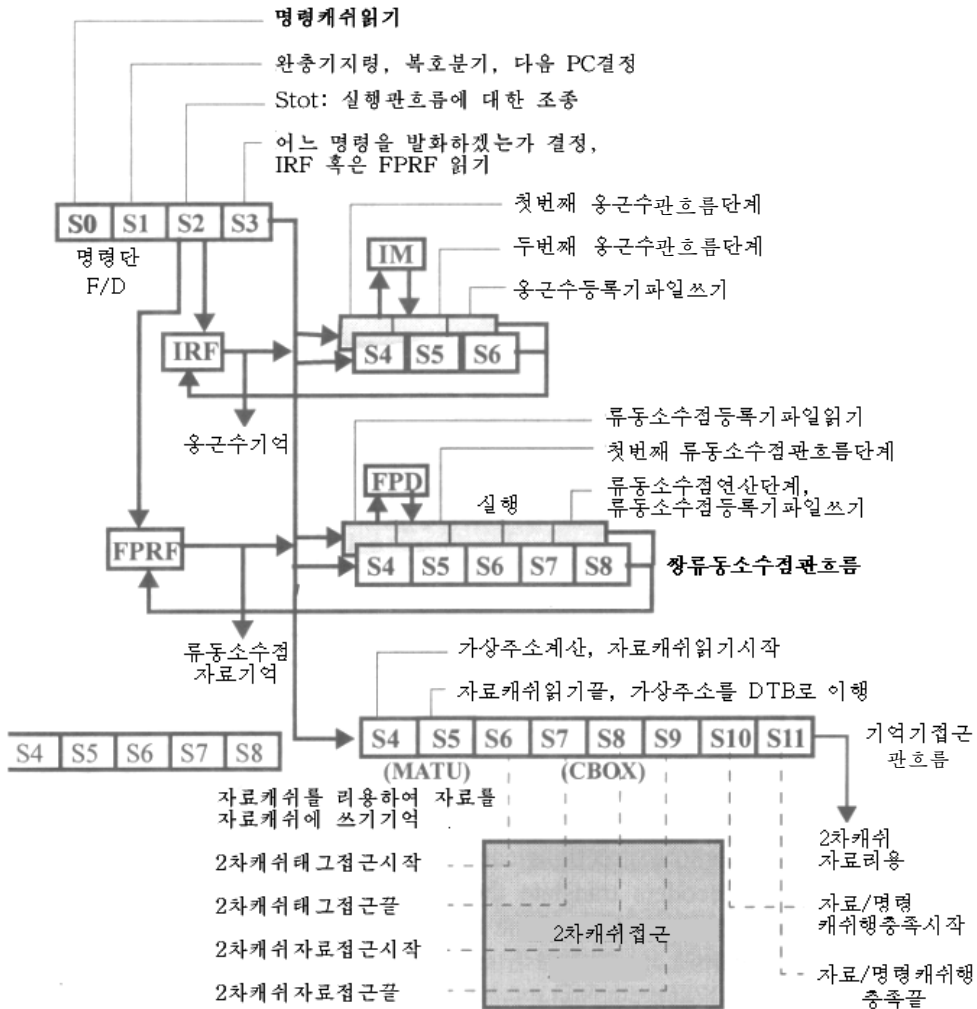



그림 4-16. 두쌍의 옹근수관흐름, 두쌍의 류동소수점관흐름 그리고 기억기접근동작을 위한 6~12 관흐름단계를 가진 21164 명령관흐름
(IRF:옹근수등록기파일, FPRF:류동소수점등록기파일, IM:옹근수곱하기, FPD:류동소수점나누기, DTB:자료이행완충기, MATU:기억기주소이행단, CBOX:캐쉬조종/모선대면부

의존형명령은 단계 S11후에 즉시 실행하기 시작할수 있다.
MATU와 CBOX는 모든 캐쉬계층준위에서 비차단방식의 캐쉬비적중을 조종한다.
이 비차단설계우점은

(1) 단번에 한개이상의 접근을 시작함으로써 쓸모있는 읽기병렬성을 개척한다.
(2) 6개까지의 읽기가 재순서화되도록 한다. 이 읽기재순서화는 쏘프트웨어관흐름과

자료미리꺼내기가능성을 준다.

다른 성능특징 21164는 동적분기예측을 실행하기 위하여 분기리력표 BHT를 리용한다. BHT는 매 분기명령의 과거 두개 실행결과를 기록하는 20482b입구자료를 가진다.

IRF에는 이름교체등록기가 없다. 그러나 8개 류동소수점등록기들은 동적실행을 지원하기 위하여 이름이 교체될수 있다.

류동소수점성능에 의해 21164는 18.3SPEC95/FP률로 되였으며 1997년에 HP PA-8000와 묶었다. 그 옹근수성능은 역시 그림 4-14에서 보여 준바와 같이 같은 성능에 귀착된다.

499-pin묶음과 함께 Alpha소편은 소편외장자료전송을 위한 128B자료경로를 가진다.

기억기대역너비는 400MB/s로 평가되였다. 에네르기소비는 21064의 50W로부터 현재 21164의 25W로 감소되였는데 주로 CMOS소편에 2.9V전력공급을 리용하였다.

명령 TLB와 자료 TLB에서 48개 입력자료들은 주소이행시간을 짧게 하도록 도와준다.

Alpha소편들은 PCs에서 리용하는것은 값이 높으므로 주로 워크스테이션에서 리용되였다.

4. 4. 2. Intel Pentium Pro 처리기

Intel Pentium Pro는 x86 CISC명령들을 직접 실행하지만 내부적으로 소편은 그림 4-17에서 보여 준 분리형CISC/RISC구성방식을 수행한다.

앞단에서 3개의 x86명령들은 순서이행기계에 의해 병렬로 복호화될수 있다. 이 복호기들은 x86명령들을 uops로 표기한 5개의 RISC계렬마이크로동작으로 넘긴다.

두개의 단순복호기들은 한개의 uop를 생성하고 일반복호기들은 복합명령을 1~4uop로 전환한다.

뒤단에서 초스칼라실행시간은 RISC핵심부의 5개 실행단에서 5개의 uops를 순서가 없이 실행할수 있다.

P_6은 5개의 실행단 즉 두개의 옹근수 ALUs와 두개의 적재/기억단, 한개의 류동소수점단을 가진다. 이 uop들은 먼저 40입력자료기록완충기(ROB)로 통과하여 요구되는 연산수들이 쓸모 있을 때까지 기억한다.

ROB로부터 uop들은 20입력자료예약장소로 발행되며 요구되는 실행이 없을 때까지 그것들을 정돈한다.

이 설계는 uop가 순서없이 실행되도록 하며 병렬실행자원들을 쉽게 리용하도록 한다.

계층기억기 Intel은 CPU소편을 가진 같은 묶음에 설치한 전용2차캐쉬소편을 설계하였다. CPU와 캐쉬소편들사이 직접 접속이 존재한다.

2차캐쉬소편은 256KB 혹은 512KB를 가질수 있다.

이 2차캐쉬는 주기당 200MHz박자로 64bit를 넘겨 준다. 소편내장캐쉬들은 16KB로 되며 명령을 위해 8KB, 그리고 자료를 위해 8KB로 갈라 진다.

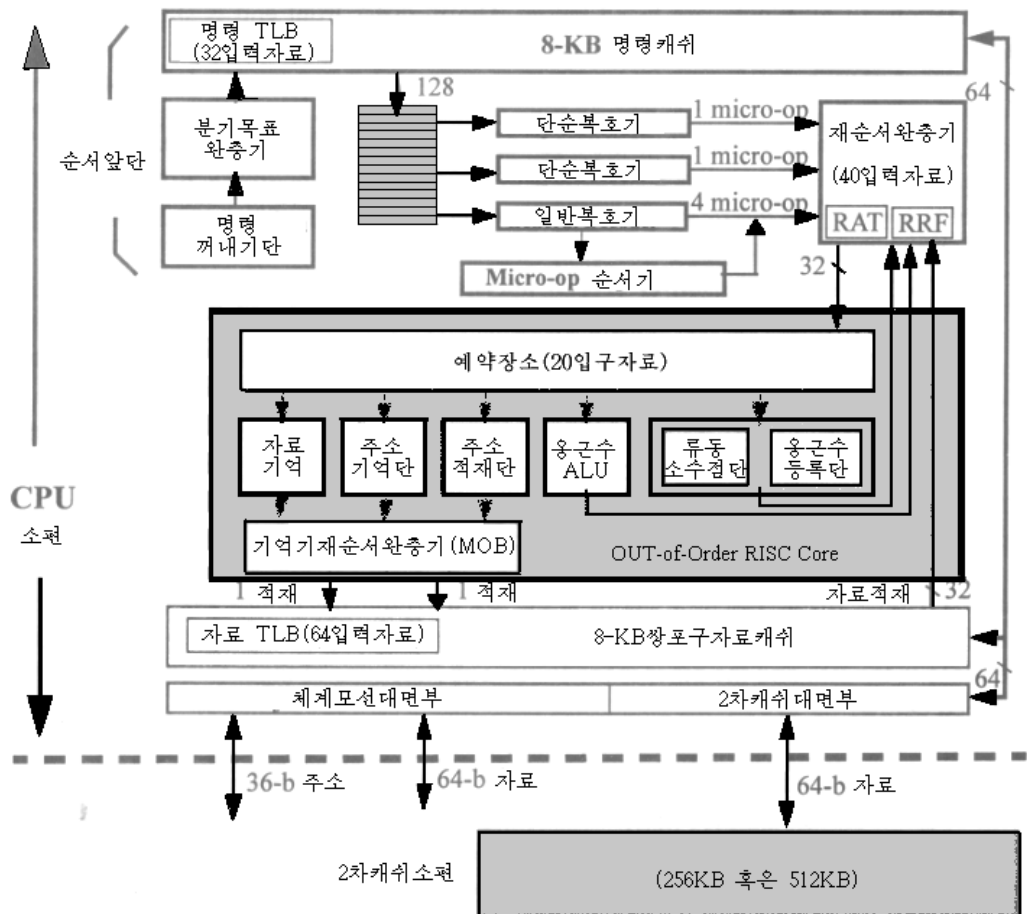



그림 4-17. Pentium Pro 처리기의 분리형 CISC/RISC 구성방식

캐쉬계층은 또한 비차단캐쉬접근을 지원하기 위해 설계된다. 두 준위사이 캐쉬통행은 뒤단 모선에서 일어 나며 앞단모선대역너비를 외부기억과 입구/출구접근에 리용되도록 한다.

순서앞단(In_Order Front End) P_6처리기에서 기능블로크들을 그림 4-18에서 보여준다.

순서앞단은 그림의 웃부분에서 보여 준다. 이 블로크들의 기능을 아래와 같이 서술한다.

- **명령꺼내기단(IFU)** I-캐쉬를 포함하여 들어 오는 x86명령들을 유지한다. 그것은 명령의 캐쉬행을 단번에 명령복호기(ID)에 공급한다. 마이크로명령순서결정장치(MIS)는 매 복합명령당 uop의 순서발생을 조절한다.
- **분기목표완충기(BTB)** 분기리력에 기초한 분기목표주소를 예측함으로써 다음명령을 꺼낸다. 512입력자료들에 대하여 2-준위적응분기도식은 련상기억기를 리용

하여 수행되였다. 분기예측오유는 투기적실행을 가진 최소값으로 감소된다.

- **등록기가명표(RAT)** 프로그람작성자가 눈에 보이는 등록기참조를 동작시간에 진행하는 내부물리적등록기들로 다시 고쳐 짓는다. Uop들은 RAT넘기기에 의해 뒤단RISC핵심부로 보낸다.

비순서핵심(Out_of_Order Core) RISC Core는 그림 4-18에서 보여 준다.
RAT는 uop들을 각이한 블로크도식아래에서 각이한 장소로 보낸다.
한 장소는 RS이고 다른 장소는 ROB이다. 이 두 기구의 기능을 아래에 서술한다.

- ROB는 재순서화가 진행되고 원천프로그람순서가 기억되는 장소이다. 프로그람순서에서 완료된 명령을 뽑아 내야 하기때문에 그것들의 원래 순서를 ROB에 기억해야 한다. 완료된 uop들이 실행단에서 돌아 올 때 오른쪽 장소에 뽑아 놓기전에 그것이 속하는 ROB로 검사한다.

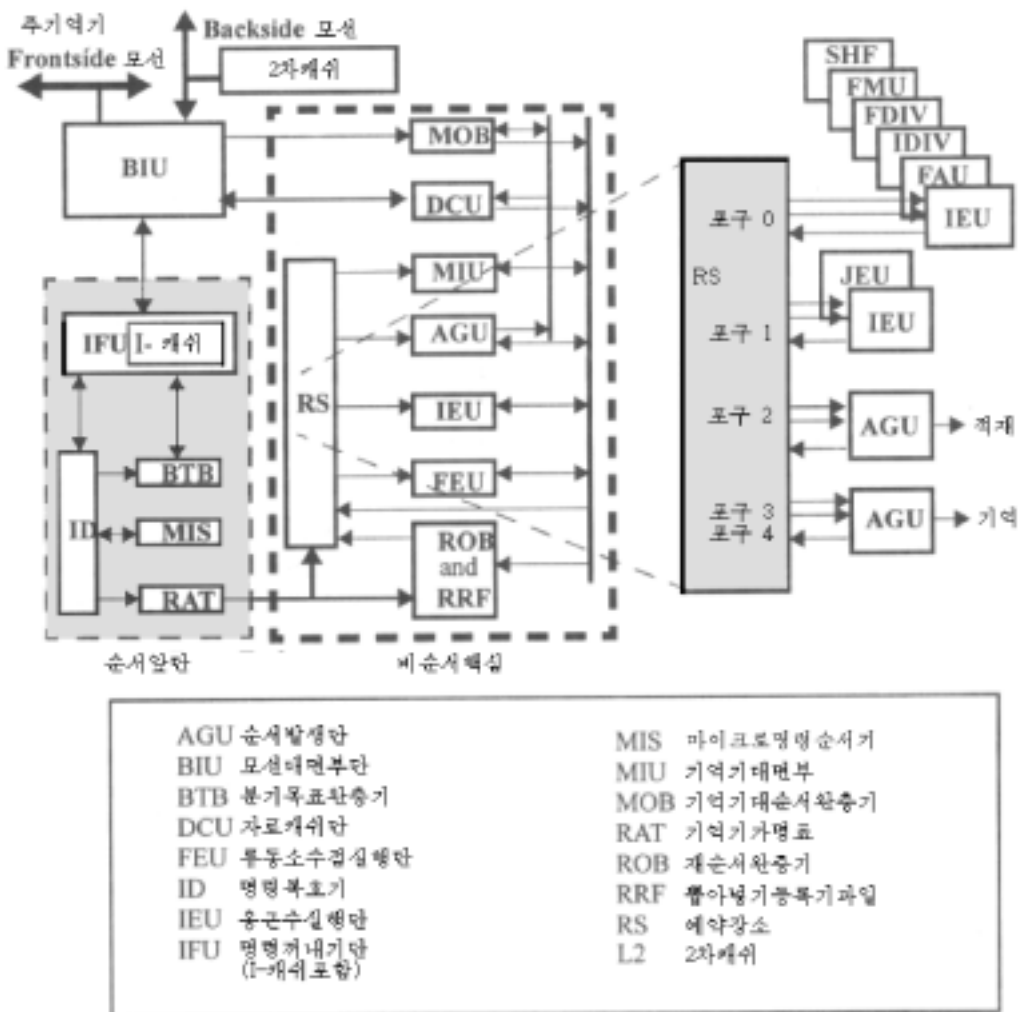



그림 4-18. 동적실행을 위한 Pentium Pro처리기마이크로구성방식

- RS는 20개 입력자료들을 가지며 매개는 실행을 위해 대기하는 한개의 uop를 유지한다. RS는 first_in, first_out(FIFO)대기가 아니다. 그것은 옹근수, 류동소수점 그리고 기발상태를 조종할수 있는데 어느것이나 다 renaminh을 허용한다. RS는 오른쪽에 그려 진 5개의 포구를 가진다.
- 포구 O는 옹근수실행단, 류동소수점가산기, 옹근수나누기, 류동소수점나누기, 류동소수점곱하기와 밀기장치에 붙인다. 포구 2, 3, 4는 기억기접근에 쓴다. 포구 2는 주소발생단(AGL)을 통하여 적재주소를 발생하며 포구 3과 4는 기억주소를 발생한다.

토막화된 초관흐름설계 Pentium Pro는 최대 14개 처리단계를 가지도록 하기 위해 관흐름화된다. 이것을 그림 4-19에서 보여 준다.

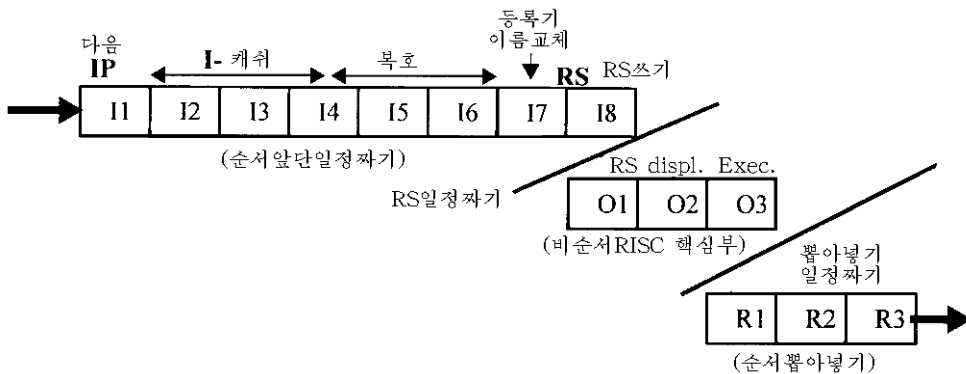



그림 4-19. Pentium Pro에서 14개 관흐름단계들의 토막화

RS동작의 융통성에 조화되도록 하기 위해 관흐름은 고정된 단계수를 가지지 않지만 명령완료의 최대박자주기수는 12로 한다.

관흐름단계들은 순서앞단 (8개 I단계), 비순서 RISC core(3개 O단계)와 순서뽑아놓기(Retirement)(3개 R단계)에서 처리기자원에 대응하는 3개의 토막으로 나눈다.

관흐름을 통한 uops의 흐름은 왼쪽에서 오른쪽으로 그려 진다.

토막들사이의 자르기들은 RS 혹은 ROB자원들을 일정작성기하기 위한 대기효과를 보여 준다. 단계의 동작들은 왼쪽에서 오른쪽으로 진행되며 아래에서 구체적으로 서술한다.

관흐름단계 I8은 일부 마이크로동작을 위한 단계 O1과 중복될수 있다. 류사하게 중복된 동작은 단계 O3과 단계 R1사이에서 일어 날수 있다. 그것은 명령이 통과하여야 하는 최소단계수가 12주기이기 때문이다. 14개 단계를 단계적으로 3개의 짧은 관흐름으로 토막화하는것은 토막들사이의 어떤 완충트랜잭션을 제공한다.

그것은 긴 관흐름을 리용할 때 생기는 심한 벌칙을 피할수 있게 한다.

- **순서복호/이행** 다음 명령지적자(IP)는 단계 I1에서 갈라 진다. 여기서 BTB는 리력에 기초하여 분기방향을 예측한다. 캐쉬접근과 명령부호화는 매개가 2.5박자주

기로 갈라 지며 3개 복호기와 명령정렬을 포함한다. 등록기이름교체단계는 가상 등록기를 물리등록기로 넘기는데 쓰인다. 마지막단계(I8)도 RS를 쓰기 위한것이다. O_1단계와 중복될수 있다.

- **비순서5-통로실행** 단계 O_1과 O_2은 RS이동을 위한것이다. 대기주기에 따라 연산수가 쉽게 리용가능하면 uop는 이 목적을 위해 1주기로만 감소할수 있다. 세번째 단계는 사실상 대부분의 마크로동작들이 1주기실행을 위한것이다. 기억기 혹은 류동소수점실행들은 많은 주기들을 걸쳐야 한다. 기억기접근은 완전히 다른 관흐름화된 동작들이며 그림 4-19에서는 보여 주지 않는다.
- **순서뽑아놓기** 뽑아놓기처리는 같은 x86명령과 관련된 모든 uop들을 한개 묶음으로 뽑아 놓게 한다. 더우기 원래프로그람순서를 복귀시킨다. 결과는 일부 uops들을 RS에서 대기하도록 하기 위해 RS로 거꾸로 흘러 가게 해야 한다. 이것은 이 모든것들을 정확히 수행하는데 3주기가 걸리기때문이다.

동적실행 Pentium Pro구성방식의 가장 중요한 특징은 이전의 X86계렬과의 2중적인 호환성이다.

소편은 사용자코드에서 고수준병렬성을 개발하기 위하여 동적실행을 해야 한다.

방법은 프로그람흐름예측에 의한 명령실행에서의 순서화와 우선권에서의 투기적인 실행명령의 최량조정을 달성하는것이다.

투기실행은 표준실행순서에서 앞에 있는 실행을 먼저 출발시킴으로써 이루어 진다.

이 투기적인 결과는 프로그람이 예측방향으로 흐르지 않는다면 버릴수 있기때문에 ROB에 일시적으로 기억된다.

뽑아놓기동안에 이 투기적인 결과들을 일시적인 기억으로부터 영구적인 구성방식상태에 기록해 둔다.

다중처리지원 4개의 P_6CPU_5와 고속 I/O통로는 성능을 잃음이 없이 같은 앞단 P_6모선과 련결될수 있다. 사실상 4-CPU4중 SHV판들은 지금 상업적생산물로서 가치가 있다. 이 P_6모선은 캐쉬일관성규약과 붙혀 진 처리기들사이에 고속동기화를 지원한다.

더우기 모선은 사용자가 I/O 혹은 망대면부장치에 붙이는 extra OEM슬로트를 제공한다.

4. 5. Post-RISC, 다매체와 VLIW

이 절에서는 극소형처리기의 최근 발전에 대하여 고찰한다.

특히 규소기술의 련속적인 발전과 명령모임구성방식을 조사한다.

또한 최근의 미소소편에 내장한 새로운 비 RISC구성방식의 특징을 검토한다. WWW와 Internet응용을 지원하기 위하여 현재극소형처리기에 내장된 다매체확장을 고찰한다.

결과 매우 높은 ILP를 개발할수 있는 미래의 극소형처리기를 제작하기 위한 VLIW 구성방식의 가능성을 평가한다.

4. 5. 1. Post-RISC 처리기의 특징

상품화된 극소형처리기들에 부가되는 보다 많은 성능특징들에는 반대되는 경향이 있다. 일부 부가된 특징들은 아직 RISC형이지만 대부분이 비 RISC 지어는 CISC형이다.

Michigan state종합대학에 있는 연구그룹[108]은 이 부가된 비 RISC특징들을 Post-RISC라고 부른다.

최근성능특징 기술이 die크기와 3극소자밀도를 높이는데로 발전하는데 따라 RISC처리기설계가들은 소편공간을 유효하게 리용하기 위한 방식을 고찰하기 시작하였다.

아래에 주요미소소편설계가들이 리용하거나 Michigan state그룹을 포함한 연구자들이 주목한 방법들을 렬거하였다.

- 등록기들을 더 첨부하고 다매체를 위한 CPU극소형구성방식을 수정
- 처리기와 같이 빨리 동작하는 소편내장캐쉬확대
- 초스칼라 혹은 VLIW실행을 위한 추가적인 기능단들을 리용
- 류동소수점연산을 가속시키기 위하여 비직결지원리용
- 관흐름깊이를 증가시키거나 혹은 토막화된 관흐름들사이의 완충화
- 적응분기예측과 회복도식
- 프로그람비순서화동적실행은 자료구동에 기초
- 앞단에서 하드웨어코드이행지원첨가
- 분기사건앞에 투기실행을 놓는다.

표 4-6 Specint Int-92**성능결과에서 경향**

회사	낡은 처리기	Spec Int-92	새로운 처리기	Spec Int-92	박자속도에서 증가률	증가률 Spec Int-92
DEC	21064A/ 300 MHz	220	21164/ 333 MHz	400	11%	82%
HP	PA-7150/ 125 MHz	136	PA-8000/ 133 MHz	360	6%	164%
IBM	PPC-6-1/ 80 MHz	91	PPC-604/ 133 MHz	176	66%	93%
Intel	Pentium 166MHz	198	Pentium Pro/ 200 MHz	320	20%	62%
Sun	Hyper-Sparc/125 MHz	131	UltraSPARC/ 200 MHz	200	12%	52%
MIPS	R-4400/ 200 MHz	141	R-10000/ 200 MHz	300	0%	113%

많은 기능적특징들이 이 목록에 첨부될수 있다.

매 미소소편계렬들이 가장 새로운 처리기를 도입하는 과정에 Michigan state그룹에서는 Specint-92와 박자속도사이의 관계에서 흥미 있는 경향성을 관측하였다. 표 4-6에서 보여 준바와 같이 Specint-92에 의해 측정된 성능은 박자속도보다 더 빨리 증가한다. 현재 처리기를 리력자료를 리용하는 먼저것들과 비교하여야 하기때문에 그룹은 Specint-92를 선택하였다.

Post-RISC특징들

Michigan state그룹은 다음과 같은 Post-RISC특징들을 제기하였는데 그림 4-20에 관흐름설계로 보여 준다.

(1) ISA는 더 증대되였다. 성능을 높이기 위해 일부 비 RISC명령들이 도입되였다. 가장 중요한 변화는 명령들을 순서화하는 지능콤파일러리용과 다중발행들의 합법성을 검사하는 하드웨어리용에서 나타났다.

(2) Post-RISC처리기들은 주기당 많은 명령들을 발행하는 령역에서 아주 강하다. 이것은 명령재순서화를 동적으로 수행하는 하드웨어를 리용함으로써 초보적으로 실현된다. 높은 ILP는 명령들을 프로그람비순서로 실행함으로써 얻을수 있다.

(3) 비순서화실행은 계산에서 새로운 개념은 아니지만 (그것은 20년동안 IBM과 CDC콤퓨터에서 존재하였다.) 단일소편실행에 대해서는 혁신적이다. 결과는 자료흐름실행과 비슷한 실행핵심부를 가진 RISC ISA이다.

그러나 이 처리기들은 아직 대부분의 RISC개념을 쓰고 있다.

실례로 이 처리기들의 실행단은 단일주기에서 대부분의 명령들을 완성하도록 최량화되였다.

(4) Post-RISC에서 새로운 구성요소는 그것의 결과를 캐쉬에서 명령과 함께 기억하는 사전복호단, 출구와 반종속을 제거하는 이름교체등록기들의 리용, 명령재순서완충기 그리고 뽑아 놓기단을 포함한다.

Michigan state Post-RISC기계 Michigan state그룹은 일부 좋은 특징들을 우에서 본 CPU설계와 합쳐 개념적Post-RISC처리기모형을 제기하였다. 다시 말하면 이것은 우에서 토론된 특징들의 혼합이다. 이 관흐름모형을 그림 4-20에서 보여 준다.

실행단전과 후에 사전복호, 재순서 그리고 완성완충기를 리용하는것은 자료흐름의 견해에 의해 활기를 띠게 되였다. 그러나 Michigan state그룹은 이 설계가 자료구동형기계들이 범하기 쉬운 오유를 피할수 있다고 주장한다.

상품화된 처리기들에서 Post-RISC특징 더 많은 Post-RISC특징을 가진 차수를 증가시켜서 Digital Alpha 21164는 최소의 RISC설계를 선택하였으며 다음절에서 연구된바와 같이 순서명령만을 실행하였다.

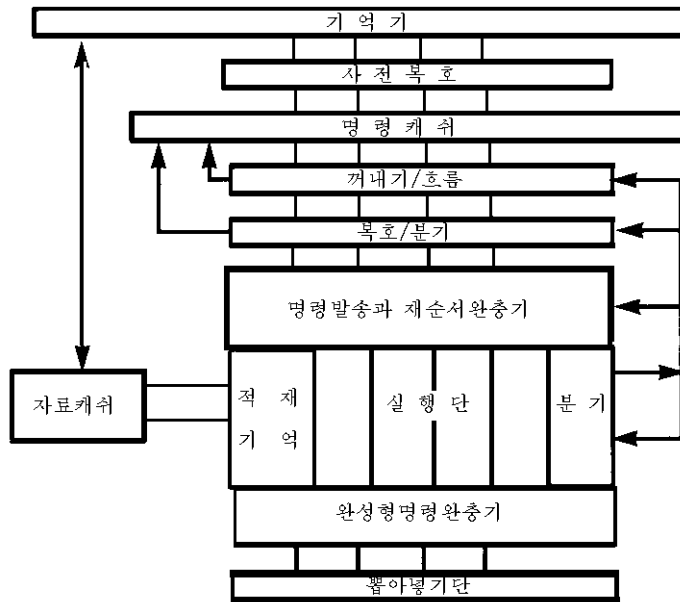



그림 4-20. Michigan state의 Post-RISC처리기에서 개념적관흐름구성방식

P_6은 우에서 고찰한 거의 모든 특징들을 가짐으로써 가장 적극적인 변화를 가져 왔다. 나머지 4개 처리기들은 두 극단들사이에 있다.

Ultral SPARC계렬 Sun Ultra SPARC는 일반적으로 비순서실행을 허용하지 않는다. 그러나 일정한 다중주기명령들은 비순서명령을 뽑아 내는것이 허용된다.

명령완충화는 두개 가상관흐름사이에서 계단적으로 보장된다.

두 관흐름의 어느 하나가 멎으면 다른 하나는 P_6에서 R와 ROB의 완충역할과 류사하게 완충기를 조사할수 있다.

Ultra SPARC처리기들은 등록기이름교체에 대한 지원이 없이 등록기창문을 리용한다. 이 방법 역시 비순서실행으로부터 소편을 막는다(보호한다.). Ultra SPARC강화는 등록기창, 변수크기페지, 고속함정(trap)과 새치기조종, 고속블로크기억기이동명령 그리고 그라픽스조작명령모임, 다매체응용을 지원하기 위한 VIS명령모임가운데서 최고속문맥절환에 달려 있다.

Ultra SPARC계렬에서 이 모든 좋은 특징들은 저비용탁상용PC응용보다는 높은급워크스테이션와 봉사기응용을 개선하는데 의의가 있다.

IMB Power PC604 이것은 4-통로초스칼라 RISC설계이다. Michigan state그룹은 이 CPU소편을 원래 RISC특징에 대응하여 세번째로 분류하였다.

그것은 재순서완충기에서 16개까지의 명령을 재순서화하는것을 지원한다. 604는 28개 이름교체완충기를 리용하는데 8개는 분기단에서 조건등록기를 지원하며 12개는 옹근수등록기를 지원하고 12개는 옹근수등록기파일을 지원하고 8개는 류동소수점등록기파일을 지원한다. 또한 많은 예약장소들은 3개의 옹근수단, 적재/기억단, 류동소수점단을 포함

하는 기능단들의 입구포구에서 리용할수 있다. 보충적으로 동적분기예측과 투기실행이 지원된다.

604에서 독특한 특징은 분기계수명령이며 계수에 기초한 고리를 수행할수 있다.

MIPS R10000 CPU는 몇개 명령역에서 우에서 본 3가지 미소소편보다 적극적이다. 그것은 32개까지의 명령을 비순서로 실행할수 있다. 옹근수연산을 위한 32개 이름교체등록기와 류동소수점연산을 위한 또 다른 32개 이름교체등록기가 있다.

특수한 특징들은 오유예측벌칙을 최소화하고 투기실행을 촉진시키기 위하여 shadow map와 분기 resume캐쉬를 리용한다.

옹근수와 류동소수점명령을 위한 64개 매개의 등록기파일의 크기는 성능을 높이기 위한 또 다른 방법이다.

R1000은 SGI/MIPS워크스테이션과 SMP봉사기에 대한 요구에 따라 그라픽스와 초고속콤퓨터리용을 지원하는데서 강한 특성을 가진다.

HP PA-8000 비순서실행에 의하여 이것은 56개 입력자료명령재순서완충기(IRB)를 리용하는 가장 적극적인 설계이다.

여기서 28개는 산수연산을 위한것이고 28개는 기억기연산을 위한것이다. PA-8000성능은 대체로 Alpha 21164성능과 비교된다.

사실상 PA-8000은 3.9×백만개의 3극소자를 론리소편에 배당하고 두번째 부류만 Pentium Pro CPU소편에 내장하는데 4.5×백만은 론리구역밖에, 5.5×백만개의 3극소자는 소편에 배당된다.

비교하면 세번째곳에 론리소편에 대한 2.4×백만개를 리용한 R10000이 있다. Power PC 604와 Ultra SPARC는 론리소편에 2×백만개의 3극소자를 리용하였고 Alpha는 론리소편에 1.8×백만개 3극소자만을 리용하고 8×백만개이상은 소편내장기억기에 리용하였다.

이 론리소편밀도는 새로운 혹은 추가된 Post-RISC특징량에 대한 좋은 지적자인데 역시 Michigan state가 정돈한 6개의 처리기와 조화된다.

4. 5. 2. 다매체확장

Internet, 다매체와 WWW리용은 현재 극소형처리기의 확장을 시작하게 하였다. 그 방법은 다매체성능을 의미있게 개선하는 명령모임을 확장하는것이다.

방법은 MPEG부호화와 복호화, 음향합성, 화상처리와 모뎀 등과 같은 시간요구함수를 추가하기 위하여 die령역에서의 작은 증가로 현재 CPU론리를 조금 변경시키는 것이다.

매체처리요구 다매체기능은 시간에 따라 증가한다. 오늘날의 탁상용콤퓨터는 계산도구외에 실시간통신과 같이 다매체정보접근을 지원하기 위해 요구된다.

다매체는 모두 수자적으로 표현된 Visual정보, 음성정보, 문자정보와 감각정보의 통

합을 의미한다.

Hewlett−Packard연구소의 Ruby Lee가 관찰한바와 같이 매체처리는 성능에서 량적인 비약과 앞으로의 콤퓨터체계의 능력을 요구함으로써 매체처리병렬성을 개척하기 위한 충분한 계기로 된다. 아래에 음성, 영상, 화상, 화소처리, 그라픽스실감묘사 그리고 변환문제를 고찰한다.

- 음성/영상처리는 16b 혹은 그이하의 정확성을 가지고 시간−의존형자료를 취급한다. 더하기−축적기는 기본연산이다.
- 많은 자료병렬성을 화소, 영상, 화상 그리고 그라픽스실감묘사프로그람에 존재한다. 자료는 작은 옹근수들(영상/화상에 대하여 8~12b)이며 중간결과들에 대해서는 16b이면 충분하다.
- 일반옹근수연산들은 더하기, 덜기, 곱하기척도 그리고 간단한 나누기를 포함한다. 기억기−접근패턴들은 흔히 영상, 화상 그리고 그라픽스연산들에서 예측할수 있다.
- 화소 혹은 선호처리는 고리벡토르화, SIMD병렬성 그리고 작은 옹근수들의 단어묶음화를 요구하며 모두 부분단어병렬성으로 이끌어 간다.
- 통신은 계산 그자체보다 더 많은 계산을 요구한다. 이것은 저비용극소형처리기에 병렬처리기술을 적용하기 위한 새로운 길을 열어 놓는다.

부분단어(Subword)SIMD/MIMD병렬성

다매체확장은 단일명령, 다중자료(SIMD)부분단어병렬성이다.

64b ALU를 취하고 자리올림사슬을 여러가지 점에서 차단되게 함으로써 본질적으로 같은 량의 론리는 두개의 32b연산, 4개의 16b연산 혹은 8개의 8b연산들을 모두 병렬로 수행할수 있다.

하나의 복잡성은 다중자리올림비트들을 쓸수 없다는것이다.

그러나 다행히도 대부분의 신호처리연산들은 포화산수에서 리득을 보았다. 자리올림비트를 회전시키고 설정하는것 대신에 포화산수는 최소 혹은 최대값에서 결과를 설정한다.

대부분의 다매체확장은 포화산수를 선택적으로 첨부한다.

다른 일반보충은 곱하기−더하기, 자료요소묶음화와 묶음해제화 등에 대한 명령들이다. 그림 4−21에서 우리는 x86구성방식을 확장하기 위한 Intel의 MMX(다매체확장)기술에서 쓰인 자료형을 보여 준다.

3개의 묶음 혹은 압축자료형을 그림 4−21 ㄹ)에서 보여 준 64b 4배단어로 정의한다.

ㄱ) 부분은 8묶음byte를 보여 주는데 매개는 8b이고 한개의 64b 4배단어로 한다. ㄴ) 부분은 64bit를 4묶음단어로 나누는것을 보여 주는데 매개는 16b이다. 류사하게 ㄷ)부분은 64b에서 2묶음 2배단어를 보여 준다. 매 묶음요소는 8b 혹은 16b 혹은 32b옹근수를 취할수 있다.

이것들은 다매체계산에서 흔히 쓰는 자료형들이다.

공업적확장 Hewlett−Packard는 먼저 MAX확장을 정확성구성방식에 추가하는것이지만

HP의 명령들은 아주 간단하다. Sun은 Ultra SPARC에서 실행된 VIS (Visual Instruction Set)에서 가장 포괄적인 확장모임을 제공한다. Sun의 확장들은 단순SIMD동작외에 화소거리와 같은 상대적인 복합명령들을 포함한다.

가장 널리 토론된 확장모임은 Intel의 MMX이며 Pentium P55C와 Pentium−2(Klamath)처리기들에서 출현하였다. 전체적으로 57MMX명령들은 x86명령구성방식(IA)에 첨가되였다.

더구나 AMD와 Cyrix 두가지는 MMX−무모순확장을 제공할것이다. Intel은 MMX−향상코드개선이 MPEG영상복호화에 대해 4배 더 좋다는것을 평가하였다.

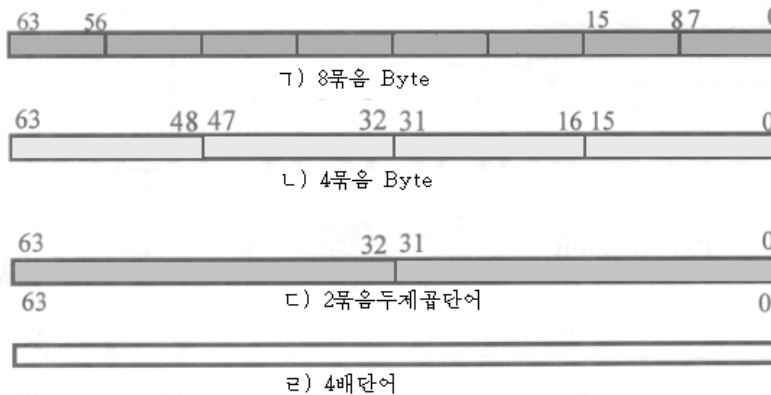



그림 4−21. Intel x86 극소형처리기에 대한 MMX 자료형

콤파일러는 MMX코드에서 락관할수 없으며 프로그람작성자들은 그 우점을 실현하기위하여 코드를 손으로 써야 한다.

Silicon그라픽스는 그것의 MIPS−V처리기를 위한 MDMX확장을 계획하였다.

Digital은 그것의 Alpha소편에 대한 MVI확장을 계획하였다.

초스칼라 ALUs의 분할 지난 시기에 초스칼라처리기들은 처리기에서 ALUs의 다중성에 비해 MIMD병렬성을 개발하는데 주로 리용되였다. 보다 적은 하드웨어변경으로 같은 ALU자원들이 SIMD병렬성을 개척하기 위해 리용되였다. 이것은 주어 진 초스칼라처리기에서 두개의 64b ALUs로 분할함으로써 론증된다(그림 4−22).

매 64b ALU는 4개의 16b ALUs로 분할되는데 그것들은 병렬로 동작할수 있다. 총체적으로 8−통로 16b산수연산들은 SIMD lock단계에서 주기당 수행될수 있다. 연산수들은 64b 다매체등록기들로부터 나오는데 매개는 4개의 16b 단어의 자료요소를 가진다.

총체적으로 16개 자료요소들은 8개 16b ALU에 의해 병렬로 가공처리된다. 실례로 다음과 같은 8개의 16b 옹근수더하기들은 모든 연산수들이 이미 등록기들에 적재되여 있다면 1주기동안에 수행된다.

8개 우연−접근 MMX등록기들은 MMX명령들에 의한 독점적인 리용을 위해 Pentium 처리기들에 내장되였다.

$$xi+yi=xi,\ i=1,\ 2,\ 3,\ 4 \quad (4.2)$$

$$yj+xj=yj,\ j=5,\ 6,\ 7,\ 8 \quad (4.3)$$

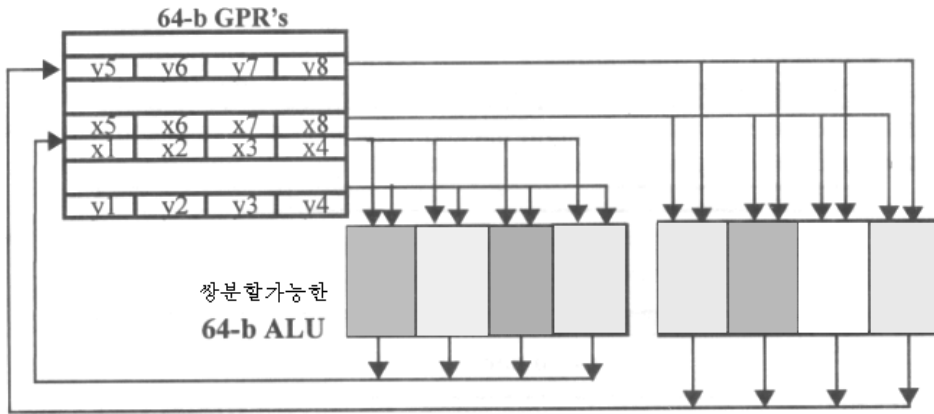



그림 4-22. 주기당 8개 16b 옹근수연산을 수행하는 초스칼라처리기에서 두개의 분할된 64b ALU리용

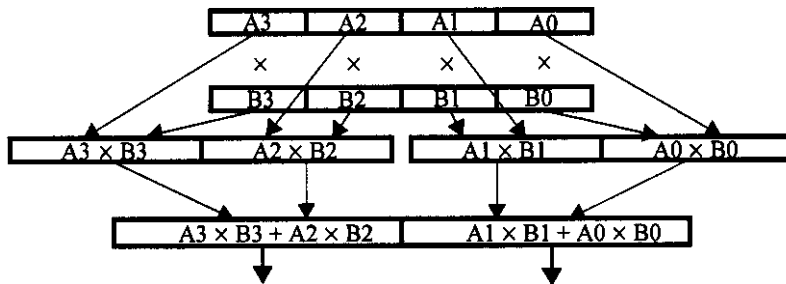



그림 4-23. 묶음화된 곱하기-더하기연산들

묶음화된 곱하기-더하기연산 그림 4-23에 자주 쓰이는 MMX연산들중에서 어느 하나로 어떻게 수행하는가를 보여 준다.

방법은 곱하기-더하기를 16b 단어에서 32b 두제곱단어들로 묶는것이다. 4개의 16b 곱하기는 묶음두제곱단어에서 4개 32b중간결과들을 얻어 내기 위해 먼저 진행된다.

두개 32b 더하기는 한개 64b MMX등록기에서 묶어 진 두개의 두제곱단어들의 32b 마지막결과를 얻기 위해 그다음에 진행된다.

6개 산수연산들의 전체 렬은 매몰형고속곱하기-축적하드웨어에 의해 가속되는바 그것은 설계에서 작은 변화만 가지고도 원래 Pentium하드웨어를 변경시킨다.

매체처리는 콤퓨터공학자들의 새로운 설계목표로 되고 있다. MAX-2확장을 가진 PA-8000에서 실행시간은 여러가지 다매체연산들에 대해 2.63에서 4.68배 속도증가에 도달하였다. 부분단어SIMD병렬성은 하드웨어와 쏘프트웨어설계에 좋은 기회를 마련해 주었다.

ISA확장외에 매체처리기 혹은 협동처리기들의 설계를 고찰할수 있다.

4. 5. 3. VLIW 구성방식

다음세대처리기를 개발하는 Intel과 HP공동개발은 VLIW방법을 리용하여 병렬성을 개발하기 시작하였다. 아래에서 지난 시기의 오유와 VLIW구성방식의 앞으로의 전망을 평가한다.

그림 4-24 ㄷ)에서 보여 주는바와 같이 다중기능단들이 VLIW처리기에서 병행으로 리용된다. 모든 기능단들은 일반 큰 등록기파일리용을 공유한다. 기능단들에 의해 동시에 실행되는 연산들은 하드웨어에 대해 동기화된다. VLIW개념은 수평마이크로코드작성으로부터 확장된다. 긴 명령의 여러가지 마당들은 발송된 여러가지 opcode를 여러가지 기능단들로 나눈다.

최대길이명령단어는 1980년 초 다중흐름콤퓨터[241]에서 설계된바와 같이 256b 혹은 1024b만큼 길수 있다.

다중흐름VLIW처리기는 극소형프로그람화된 조종에 의해 실행되였다. 256b다중흐름모형은 7개까지의 연산들이 병행하여 실행되도록 하였다. 박자속도는 극소형프로그람화된 조종을 가진 조종기억기의 빈번한 접근으로 하여 아주 높이 올라 갈수 없다.

VLIW처리기는 초스칼라처리기의 극단으로 볼수 있다. 그러한 처리기는 독립이고 비관련된 모든 연산들을 콤파일시에 벌써 함께 묶어 놓는다. RISC명령단어들(매개는 32b)로 작성된 프로그람들은 VLIW명령을 만들기 위해 밀집되여야 한다.

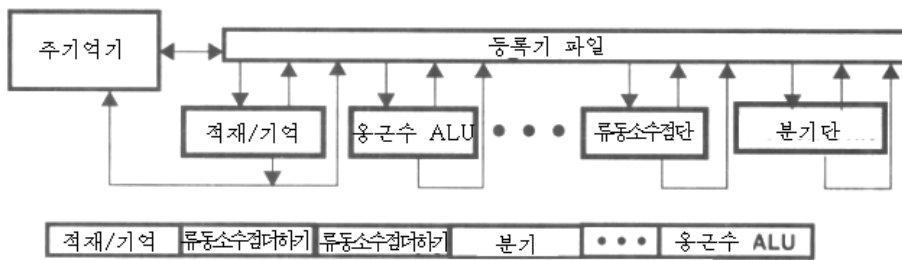
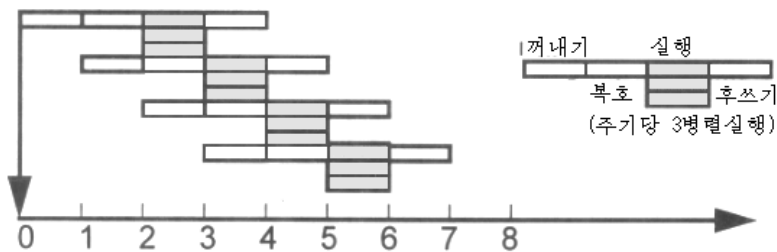
이 코드밀집은 프로그람흐름을 추적하고 추적정보를 써서 분기결과를 예측할수 있는 강력한 콤파일러에 의해 수행되여야 한다.

ㄱ) A VLIW 처리기구성방식과 명령형식

ㄴ) VLIW의 관흐름화된 실행

그림 4-24. 처리기관흐름실행단계에서 VLIW구성방식과 다중연산의 병렬실행

다중흐름기계의 오유는 주로 짧은 사용자코드단어를 VLIW명령으로 효과적으로 밀집하는 강력한 콤파일러의 비능률성에 기인된다.

P_7처리기가 VLIW구성방식을 받아 들이자면 콤파일러지원이 필요하다.

VLIW처리기우점 리상적인 VLIW처리기에서의 한 명령실행을 그림 4-24 ㄴ)에서 보여 준다. 매 명령은 다중연산들을 규정한다. 효과적인 CPI는 매 긴 단어에서 *n*-통로병렬성을 가짐으로써 1/*n*로 감소한다.

VLIW기계는 다음의 령역에서 초스칼라기계에 비해 개선된다.

(1) VLIW에서 병렬연산들의 동기화는 콤파일시에 완전히 실현되여 초스칼라처리기를 리용하는것보다 더 높은 처리기효률성을 가지게 한다.
(2) VLIW에서 리용가능한 ILP이 짧은-형식사용자코드에서 높을 때 코드길이는 훨씬 짧다. 이것은 콤파일된 VLIW프로그람의 실행시간이 훨씬 짧다는것을 의미한다.
(3) VLIW명령에서 명령병렬성과 자료이동은 실행시에 완전히 규정되기때문에 실행시간자원일정작성기는 훨씬 간단해 진다.

VLIW Pitfalls(함정) VLIW구성방식의 스칼라연산가운데서 우연병렬성은 벡토르초고속콤퓨터에서 개발된 정규병렬성 혹은 SIMD콤퓨터에서의 lock step병렬성대신에 개발된다.

VLIW처리기는 지능콤파일러에 의한 효과적인 코드실행이 없이는 리용될수 없다.

VLIW구성방식은 오늘날의 x86 혹은 RISC프로그람과 일치하지 않는다. 이러한 리유로 HP/Intel공동개발은 뒤방향쏘프트웨어혼합성론점을 취급해야 한다.

VLIW에 기초한 구성방식의 성공은 하드웨어로부터 일정작성기함수를 접수하는 효률적인 콤파일러에 크게 달려 있다.

다른 문제는 어느 한 VLIW기계를 위해 콤파일된 쏘프트웨어는 다른 VLIW기계를 위해 재콤파일되여야 한다는것이다.

다른 말로 ILP가 VLIW기계들의 세대들사이에서 증가될 때 새로운 콤파일러가 개발되여야 한다.

이 모든것들은 VLIW가 지난 시기 상업적성공이 이루어 지는데 난관을 조성하였다.

4. 6. 극소형처리기의 미래

미래 극소형처리기소편들은 밀도가 훨씬 높은 미소소편, 높은 박자속도, 높은 ILP, 낮은 CPI, 소비전력증가와 보다 정교한 쏘프트웨어지원으로 설계될것이다. 이 절에서는 미래의 극소형처리기에 대한 하드웨어, 작업부하, 구성방식, 쏘프트웨어지원에서의 추세를 평가한다.

자료는 IEEE Micro, 1996.12과 IEEE Computer Magazin, 1997.9에 제출된 현대극소형처리기들에 대한 두가지 전문론점에 기초한다.

아래에 미래 극소형처리기에 대한 일반표상을 준다.

적용범위는 21세기까지 확장될것이다.

어느 구성방식개념이 성공적인 상업제품으로 될수 있는가 하는것은 많은 기술적, 업무적인자들에 달려 있다. 일부 인자들은 아래에서 상세히 서술한다.

4. 6. 1. 하드웨어의 추세와 물리적한계

1994년에 반도체공업협회(SIA)는 2010년에 가서는 800×백만 3극소자 CPU소편들은 수천개의 Pin, 10000b모선, 박자속도는 2GHz이상, 전력소비는 180W이상의 특징을 가지게 될것이라고 예측하였다.

다른말로 10억개의 3극소자로 구성된 극소형처리기들이 다음 10년에 나타날것이다. 특히 고성능극소형처리기들을 생산하는 이전의 경험에 기초하여 Intel과 Digital의 미래에 대한 평가를 보기로 한다.

Intel개발계획 Intel극소형처리기생산품의 책임자인 Albert Yu는 1996년 말에 미래의 극소형처리기를 예측하였는데 표 4-7에 요약되여 있다. 그 예측을 보면 2006년에 3극소자 총수가 350×백만으로 뛰여 올라 갈것이라고 하였다.

die크기는 매면에서 0.7in에서 14in으로 증가할것이다. 이 예측은 오히려 보수적이다. 왜냐하면 0.1μm기술은 단일소편에 1조개의 3극소자를 집적할수 있기때문이다.

표 4-7 미래의 극소형처리기특징들의 예측

특 징	1996 (현재)	예 측	
		2000	2006
박자속도	200	900	4000
3극소자	6	40	350
소편크기* (in)	0.7	1.1	1.4
선폭(μm)	0.35	0.2	0.1
성능: MIPS SPEC int 95	 400 10	 2400 60	 20,000 500

* 4각형소편의 한변의 길이

박자속도가 900MHz로 증가한다는것도 완전히 보수적이다. 그러나 10년동안 4GHz까지 증가한다는것은 아주 놀랍다. 이것은 미래의 미소소편에 더 많은 가능성을 부여할 기회가 존재한다는것을 의미한다.

극소형처리기성능은 이 개발계획에 따라 10년동안에 50배로 증가할것이다.

DEC개발계획 Digital의 공학자들은 2003년까지 Alpha CPU소편의 전망을 표 4-8에서와 같이 평가하였다.

Digital의 개발계획은 Intel과 조금 차이난다.

5년동안 박자속도는 두배로 될것이다. DEC는 2000년까지는 명령발행률을 주기당 16명령으로 증가하고 2003년까지는 32-통로병렬성으로 증가할것으로 본다.

표 4-8 Digital Alpha CPU소편전망

모형-년도	21264(EV6) 1997	EV7 2000	EV8 2003
CMOS 처리 (μm)	0.35	0.25	0.18
소편밀도 (million)	15	100	250
박자속도 (MHz)	500	750	1000
명령발행률	8-way	16-way	32-way
SPEC int92(projection)	800-1000	3000	Unknown

ILP의 그러한 높은 수준을 구성방식과 콤파일령역에서 기본돌파가 없이 개발하는것은 오히려 어렵다.

앞으로 웨이퍼기판크기의 확대, 얇은 선폭을 0.1μm로 낮추는 기술이 예상된다.

그러나 die크기에서 확대, 얇은 선폭, 아주 높은 박자속도는 새로운 물리적제한을 만들어 내는데 아래에 서술한다.

물리적제한 Martzke[435]는 가장 중요한 물리적제한으로 선폭과 직결장치들이 앞으로 줄어듬에 따라 직결선이 론리문보다 훨씬 떠진다는 사실을 지적했다.

단일대역박자는 전체 미소소편에 있는 10억개의 3극소자를 구동할수 없다. 앞으로 크기가 작아 진다는것은 좋은 소식이지만 확대가능성, 시간(skewing)은 좋아 지지 않을것이다.

Die의 16%만이 10억개의 3극소자처리기에서 단일박자주기안에 도달할수 있다. 초소편을 통해 신호를 보내는것은 1.2GHz박자의 20개 주기를 요구할수 있다. 다른 제한은 큰 CPU소편에서 방출되는 과도한 열로 하여 생긴다. 랭각과 묶음화는 그다음의 현실적문제이다.

4. 6. 2. 미래의 작업부하와 난관들

처리기구성방식은 예상된 응용작업부하에 의해 구동된다.

일반목적과 응용특정처리기 두가지는 다음 20년기간에 작업부하에서 기본변화를 가져 올것이다. 실례로 인터네트와 WWW리용은 극소형처리기들에서의 다매체와 통신확장을 촉진하였다.

사용자대면부는 다매체극소형처리기에서 더 많은 전력을 소비할것이다.

다매체작업부하는 실시간과 매몰형응용에서 계속 증가할것이다.

HP의 Ruby Lee는 부분단어병렬성을 매체처리기구성방식, 캐쉬와 기억기체계, 부분단과 병렬성, 콤파일러, 그림언어와 응용알고리듬에서 혁신적인 좋은 기회를 제공한다는것을 지적하였다.

추가적으로 설계, 확인, 극소형처리기검사는 전체 개발비용의 40%이상을 차지한다. 조립공장은 고성능극소형처리기를 대량적으로 생산하는데 쉽게 2조딸라를 초과할수 있다.

다음세대처리기를 개발하기 위한 Intel/HP P_7의 목적은 분리형구성방식의 우수한 특

징들을 VLIW와 다매체구성방식과 통합하는것이다. 구성방식 EV7, EV8 그리고 P_7은 앞으로 중요한 목표로 된다.

하드웨어진보외에 미래의 극소형처리기들에서 다른 난관들에는 특정작업부하를 위한 새로운 처리기들에서 ILP개발을 포함한다.

콤파일과 실행시간체계쏘프트웨어는 미래의 구성방식에서 병렬성을 자동적으로 얻기 위해 요구된다.

4．6．3．미래의 극소형처리기구성방식

미래의 조 1-3극소자극소형처리기에 대해 Berger와 Goodman[115]은 그림 4-25에서 7개의 후보구성방식을 보여 준다.

구성방식이 아래로 내려 갈수록 그것들은 현재 프로그람작성모형과 멀어 진다.

꼭대기에 있는 현재 극소형처리기들은 평가경로에 필요하다.

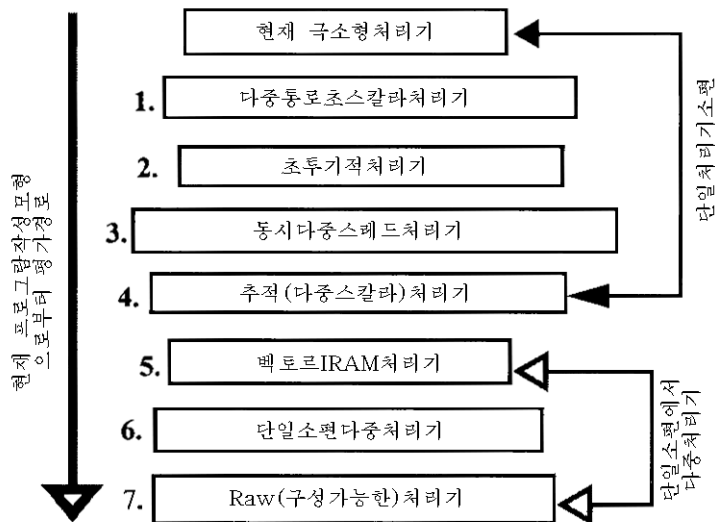



그림 4-25. 미래의 처리기구성방식의 개발평가

이 구성방식추세는 수직축을 따라 순서화된다.

앞절들에서 토론된 이 인자들은 미래의 구성방식에 대한 실제적인 장성경로를 결정한다.

그림 4-25에 있는 8개 구성방식가운데서 현재미소소편들과 꼭대기 4개 구성방식들은 단일소편에서 단일처리기이다.

그 나머지는 다중처리기소편들이다. 현재 ISA에서 벗어 나려는 추세는 현재 프로그람작성모형을 유지하려는것과 평형되고 있다. 이러한 구성방식들은 대부분 아직도 연구단계에 있다.

그것들을 아래에서 간단히 소개한다.

(1) **다통로초스칼라처리기** 오늘의 초스칼라처리기들은 대부분 3−발행 혹은 4−발행 설계를 가지고 있다. Michigan종합대학의 연구그룹은 명령공급, 자료기억기공급 그리고 실행기구는 현재 초스칼라처리기들이 16−통로 혹은 32 −통로설계로 확대하지 못하게 하는 기본문제라는것을 밝히였다. 그들은 비순서꺼내기, 다중혼성분기예측과 명령공급을 개선하는 추적캐쉬들을 리용하려고 한다. 또한 큰 소편내장캐쉬들과 자료공급을 높이는 자료값을 리용하려고 한다. 그들은 크고 비순서발행명령창(2000개 명령), 클라스터화된 기능단들의 뱅크와 준비명령의 계층적 일정작성기를 주장하며 현재 단일처리기소편과의 쏘프트웨어호환성을 유지하려고 한다.

(2) **초스칼라처리기** Carnegie−Mellon종합대학의 연구그룹은[415] 성능개선에 대한 모든 준위에서 대량의 투기리용에 주의를 집중하였다. 그들은 여러가지 기억기구성을 위한 꺼내기폭 32, 128개의 재순서완충기, 128입력자료기억대기를 가진 초흐름마이크로구성방식을 제기하였다. ISA에 대한 재편집과 변화를 요구함이 없이 어떤 성능평가기준을 위한 주기당 19까지의 명령초투기적성능과 SPEC 95옹근수를 위한 조화평균 SIPC를 얻기 위해 약한 의존성모형을 리용한다. 이러한 노력은 많은 점에서 Michigan다통로초스칼라처리기를 보충한다.

(3) **동시다중스레드처리기** 동시다중스레드화(Simultaneous Multi Threading, SMT)는 Washington대학의 다중문맥단일처리기를 의미한다. SMT방법은 단일스레드, 단일처리기구성방식에서 벗어 난다. SMT처리기는 다중프로그람화된 작업부하밖의 다중스레드가운데서 적극적인 관흐름을 공유한다. 이 방법의 성공은 스레드준위에서 높은 ILP의 유용성에 크게 달려 있다. 지금까지 모의결과만이 이 방법의 일부 성능리득을 보여 주었다.

(4) **추적(다중스칼라)처리기** 이 개념은 Wisconsin종합대학의 Smith와 Vajapeyam에 의해 제기되였다. 기본내용은 여러가지 추적코드를 동시에 실행하는 다중직결처리기핵심부들로 이루어 진 추적처리기를 리용하는것이다. 추적을 선택하는 분기예측을 리용하여 한개 핵심부외의 모든 핵심부가 투기적으로 실행된다. Wisconsin그룹은 미래의 다중스칼라처리기들이 보통 순차프로그람의 실행속도를 증가시키는 복제, 계층성 그리고 예측에 크게 의거할것이라고 주장한다. 실례로 매 처리요소가 주기당 4개 명령들을 발행하면 4개의 처리요소는 주기당 16 혹은 그이상의 명령들을 실행할수 있다.

(5) **벡토르 IRAM처리기** IRAM(Intelligent Random−Access Memory)는 California종합대학의 연구개발계획이다[377]. 그들은 기억기체계가 앞으로 주요한 성능병목문제일것이라고 주장한다. 그들은 소편우의 대용량기억기배렬안에 내장된 확대가능한 다중처리기를 제작하는 DRAM기술을 조사하고 있다. 결과 소편내장기억기용량과 높은 대역너비는 일반적인 구성방식보다 비용이 효과적인 벡토르처리기들이 훨씬 높은 성능준위에 도달할수 있다고 본다. 그 방법은 현재 초고속콤퓨터에서 충분히 만족되기때문에 명백한 편집을 요구한다. 그들은 미래의 작업부하는 보다 벡토르화가능한 성분들을 포함할것으로 믿고 있다.

(6) **단일소편다중처리기** 이것은 소편다중처리기(CMP)를 발전시킨 Stanford종합대학

의 개발계획인데 한개 소편우에 4~16개의 고속처리기를 내장하였다. 매 처리기는 작은 1차캐쉬에 밀집으로 결합되고 모든 처리기들은 큰 2차캐쉬를 공유한다. 처리기들은 병렬일감을 협조하거나 독립과제를 수행할수 있다. CMP를 잘 동작하게 하기 위해서 프로그람작성자나 콤파일러는 코드를 명백히 병렬로 작성하여야 한다. 이전의 ISA는 비록 한개의 작은 처리기에서 천천히 혹은 비효률적으로 동작할수 있지만 CMP와는 적합하지 않을것이다. Stanford연구자들은 스레드와 처리병렬성을 앞으로 널리 퍼지게 되리라고 기대하고 있다.

(7) **Raw(구성가능한)처리기** MIT연구소의 이 개발계획은 가장 기본적인 구성방식을 제공하는데 우리가 지금까지 연구한 일반적인 구성방식 혹은 미래의 구성방식에서 떨어 져 있다. 기본내용은 매개가 단일소편에서 재구성가능한 론리를 가진 수백개의 매우 간단한 처리기들로 높은 병렬구성방식을 완성한다. 병렬실행과 통신은 쏘프트웨어조종과 조정으로 재분류한다. 이 방법은 전통적인 명령모임대면부를 배제한다. 그것은 복제된 구성방식을 직접 콤파일러에게 로출시킨다. 이 구성가능한 구성방식은 콤파일러가 매 응용프로그람에 대한 하드웨어를 구분하도록 한다.

4. 7. 참고문헌주해와 련습문제

발전된 극소형처리기들이 [603]에서 고찰된다. Crawford는 [173]에서 i486구성방식을 서술한다.

M6840처리기는 Ecenfield에 의해 [224]에서 서술된다.

RISC콤퓨터에 대한 리론은 Stalling[581]에서 주어 진다.

RISC구성방식에 대한것은 Hennessy와 PatterSon[305]에 서술되여 있다.

처리기를 취급하는 Hwang의 책은 모든 콤퓨터교재에서 쓰이고 있다. 가장 초기의 극소형처리기인 Intel4004의 력사가 Faggin 등에 의해 고찰되였다[234]. Berkeley RISC는 Patterson과 Sewuin에 의해 제기되였다. MIPS R2000은 [359]에서 토론되였고 MIPS R4000은 Mirapuri 등에서 보고되였으며 [456], MIPS R10000는 Gwennap[291]과 Yeager[660]에서 고찰되였다.

초스칼라 그리고 초관흐름화된 기계들은 Jouppi와 Wall에 의해 특징화되였다[356]. Johnson[352]는 초스칼라극소형처리기의 대부분의 설계론점을 취급하였다.

SPARC구성방식은 Weaver와 Germond[637]에서 특징화되였다. SPARC에 대한 콤파일러는 Munchnick에서 출현하였다[456]. UltraSPARC는 Greenley 등에서 취급되였다[281], [617]. Power PC601에 대한 다중처리기지원은 [23]에서 고찰되였다.

Power PC604는 Denman 등에 의해 서술되였다[202], [578]. Power PC620은 Levitan 등에 의해 서술되였다[408].

HP구성방식은 Lee에 의해 평가되였고[398] PA−8000은 Hunt에 의해 서술되였다[322].

MPPs에서 극소형처리기를 리용하는것은 Smith에 의해 평가되였다[571].

Alpha AXP구성방식은 Digital manuals에서 서술되였다[191], [196], [225]. IBM에서 RISC처리기발전은 [160], [335]에서 제출되였다.

Pentium구성방식은 Anderson과 Shanley[31]에서 서술되였다.

I860구성방식의 중요한 내용은 Margulis[428]에서 서술되였다.

Pentium Pro(P_6)구성방식은 Colwell과 Steck[162] 그리고 Gwennap[290]에서 서술되였다. 극소형처리기의 다매체확장은 Lee[400], [401], Peleg[445] 그리고 Weiser[640]에서 찾아 볼수 있다. Dicfendorff와 Dubey[205]은 다매체작업부하가 극소형처리기설계를 어떻게 변화시킬수 있는가를 평가하였다.

VLIW구성방식은 처음으로 Fisher[241]에 의해 제기되였다.

Post−RISC자료는 Brehob 등의 보고에 기초하고 있다[108].

[321]은 처리기소편에 대한 가장 최근의 발전을 제시한다.

극소형처리기의 사회에 대한 영향은 Markoff[429]에서 토론되였다.

미래의 극소형처리기들의 평가는 Yu[663]에서 보여 준다.

1조−3극소자구성방식은 Burger와 Goodman에서 고찰된다[115]. Matzke[435]는 처리기확대가능성에 대한 물리적제한을 평가한다.

17개의 미래의 극소형처리기구성방식은 Patt 등에서 제기되였다[491], [418].

Smith와 Vajapeyam[572], Eggers 등[227], Kozyrakis 등[377], Hammond 등[296] 그리고 Waingold[631].

문 제

문제 4.1. 처리기기술에 관계되는 다음의 기본용어들을 정의하시오.

- 명령발행률
- 주소화방식
- 단일캐쉬 대 분할캐쉬
- 하드웨어구성된 조종 대 극소형프로그람화된 조종

문제 4.2. 스칼라 RISC 혹은 초스칼라 RISC처리기에 대하여 다음질문에 대답하시오.

- 왜 대부분의 RISC옹근수단은 32개의 일반목적등록기를 사용하는가? SPRAC구성방식에서 수행된 등록기창개념을 설명하시오.
- 큰 등록기파일과 큰 D−캐쉬사이의 설계절충은 무엇인가?
- 스칼라 혹은 초스칼라조직을 가진 대부분의 RISC처리기들에서 옹근수단과 류동소수점단사이의 관계는 무엇인가?

문제 4.3. 4.2절에서 고찰한 발전된 처리기들의 토론에 기초하여 RISC, CISC, 초스칼라와 VLIW구성방식에 대한 다음의것들에 대답하시오.

- 명령형식, 주소화방식 그리고 명령당 주기에 의해 RISC와 CISC처리기들에서 명령모임구성방식을 비교하시오.
- 명령과 모임에 대한 일반캐쉬 혹은 분할캐쉬를 리용하는 우점과 결점을 토론하시오.

- 하드웨어요구와 쏘프트웨어지원에 의해 초스칼라와 VLIW구성방식사이의 차이를 설명하시오.

문제 4.4. 다음의 5개 극소형처리기구성방식을 실례들어 특징화하고 매 경우에 극소형구성방식을 연구하시오.

- 비순서실행을 가진 초스칼라 RISC
- 다매체확장을 가진 Post−RISC
- 초투기적극소형구성방식
- IRAM 혹은 지능RAM구성방식
- 단일−소편다중처리기(CMP)

문제 4.5. 상품화된 극소형처리기들의 다매체확장에 대하여 다음의 두가지 질문에 대답하시오.

- 오늘날의 64b 초스칼라 RISC처리기에서 SIMD병렬성을 개발할수 있는 주요다매체자료형들은 무엇인가.
- 64b, 4−발행인 초스칼라처리기를 리용할 때 최대속도는 얼마인가. 초스칼라처리기는 묶음 Byte SIMD/MIMD다매체처리를 위한 4개 옹근수 ALUs를 가진다. 처리기는 다매체확장을 가진다고 가정한다. 다매체확장이 없는 동일한 처리기에 대해 모두 비교하시오.

문제 4.6. 40MHz처리기에서 200000명령을 가진 대상코드실행을 고찰하시오. 프로그람은 4개의 주요명령형태로 이루어 진다. 명령혼합과 매 명령형태에 필요한 CPI는 아래에 주어 지는데 프로그람추적실험에 기초한다.

- 프로그람이 우의 추적결과를 가진 단일처리기에서 실행될 때 평균 CPI를 계산하시오.
- CPI에 기초한 대응하는 MIPS를 계산하시오.

명령형태	CPI	명령혼합
산수와 론리	1	60%
캐쉬적중을 가진 적재/기억	2	18%
분기	4	12%
캐쉬비적중을 가진 기억기참조	8	10%

문제 4.7. 40MHz처리기가 다음의 명령혼합과 박자주기수를 가진 성능평가기준프로그람을 실행하는데 리용되였다. CPI, MIPS률과 이 프로그람의 실행시간을 결정하시오.

명령형태	명령계수	박자주기계수
옹근수산수	45,000	1
자료이행	32,000	2
류동소수점	15,000	2
1조종이행	8,000	2

문제 4.8. 25MHz박자속도를 가진 선형관흐름처리기에 의해 15000개 명령프로그람실행을 고찰하시오.

명령관흐름은 5개 단계를 가지며 한개 명령은 박자주기당 발행된다고 가정한다. 분기에 의한 벌칙은 ㄱ)에서 무시되고 ㄴ)에서는 무시되지 않는다.

ㄱ) 프로그람을 실행하는 이 관흐름을 리용하는데서 속도인자를 계산하시오. 프로그람은 같은 량의 흐름통과지연을 가진 등가인 비관흐름화된 처리기리용을 비교한다.

ㄴ) 25%의 보충적인 주기들이 같은 코드를 실행하는데 필요되고 분기효과를 포함한다면 이 관흐름화된 처리기의 효률성과 통과능력은 얼마인가?

문제 4.9. 주기당 *m*개 명령을 발행할수 있고 주기당 한개 명령을 발행하는 기초처리기보다 *n*배 더 빠른 박자에 의해 구동되는 초관흐름화된 초스칼라처리기를 고찰하시오.

- 기초처리기에 비한 초관흐름화된 초스칼라처리기의 속도식 *S(m, n)*을 유도하시오.
- M-발행초스칼라처리기증대를 막는 실천적제한은 무엇인가?
- N배 더 빠른 박자를 가진 초관흐름의 증대에 대한 실천적제한은 무엇인가?

문제 4.10. 비관흐름화된 처리기 X가 25MHz박자속도와 평균 CPI 4를 가진다. X의 개선된 계승인 처리기 Y가 5-단계관흐름으로 설계된다. 그러나 크로스바(latch)지연과 박자비대칭(Skew)효과로 인해 Y의 박자속도는 20MHz뿐이다.

- 100개 명령을 포함하는 프로그람이 두가지 처리기에서 실행된다면 처리기 X와 비교되는 처리기 Y의 속도는 얼마인가?

문제 4.11. 오늘날의 일부 극소형처리기들에서 Post-RISC특징을 고찰하시오.

ㄱ) 순수한 초스칼라 RISC특징과 차이나는 적어도 4개의 Post-RISC특징을 설명하시오.

ㄴ) ㄱ)에서 동일시되는 매 post-RISC특징에 대하여 그 특징을 가진 한개 처리기실례를 동일시하고 독자의 주장을 정당화하시오.

ㄷ) 다음의 10개 처리기를 ㄱ), ㄴ)에서 동일시되는 매몰형Post-RISC특징을 가진 순서로 정렬하시오.

PowerPC 620:	AMD K6:
PA-8000:	Alpha 21164:
Pentium-2:	Pentium P55c:
UltraSparc-2:	MIPS R10000:
PowerPC 604:	Alpha 21064:

제5장. 분산기억기와 지연시간허용성

현대 콤퓨터가동환경에서 기억장치(캐쉬, RAM소편, 하드디스크, 테프단 그리고 주변장치)비용은 처리기핵심부비용을 훨씬 초과하였다. 이 장에서 우리는 기억기들에서 최근 발전에 대하여 평가한다.

특히 분산기억기구성방식과 기초적인 지연시간허용기술에 중점을 둔다. 원격기억기 접근으로 초래되는 지연시간을 줄이거나 피하며 은폐시키는 하드웨어와 쏘프트웨어방법들이 연구되고 있다.

우리는 기억기공유캐쉬일관성, 기억기일치성 그리고 확대가능한 병렬콤퓨터에서 지원시간은페를 지원하기 위한 하드웨어와 쏘프트웨어기구에 관심을 돌린다. 기억기구성방식에 관한 이러한 문제들은 극소형처리기의 성능, 확대가능성 그리고 프로그람화가능성과 CC-NUMA, COMA, 쏘프트웨어실행형DSM 등과 같은 여러가지 분산기억기구성방식을 선택하는 다중콤퓨터에 영향을 준다. 여기서는 기억기단어로부터 캐쉬행과 가상페지까지의 범위에 속하는 모든 립도준위에서 기억기공유를 취급한다.

5. 1. 계층기억기기술

대표적인 계층기억기에 대한 기억기특징들과 동작성질들을 아래에서 요약하여 서술한다. 그다음 그 계층기억기의 용량계획작성을 위한 선형계획모형을 제기한다.

5. 1. 1. 기억장치의 특성

등록기, 외부캐쉬, 디스크장치 그리고 테프단들과 같은 기억장치들은 흔히 그림 5-1에서와 같은 계층성을 가지고 조직된다. M_i는 i번째 준위에서의 기억기를 가리킨다고 하자.

그림 5-1은 4개 준위를 가진 계층성을 보여 주는데 여기서 M_0은 CPU에서 등록기 혹은 내부캐쉬, M_1은 외부캐쉬, M_2은 주기억기, M_3은 디스크기억 그리고 M_4는 테프단이다. 매 준위에서 기억기기술과 조직화는 아래에서 정의된 5개 파라메터들로 특징화된다.

- 기억기 M_i에 접근하는 접근시간 $t_i(\mu s)$
- M_i의 기억기용량 s_i (MB)
- M_i의 byte c_i당 비용(cent)
- M_i와 M_{i+1}사이의 자료전송률 혹은 대역너비 b_i(MB/s)
- 린접한 준위 M_i와 M_{i+1}사이의 byte 혹은 단어, 캐쉬행, 페지 혹은 토막들에서 x_i 전송단

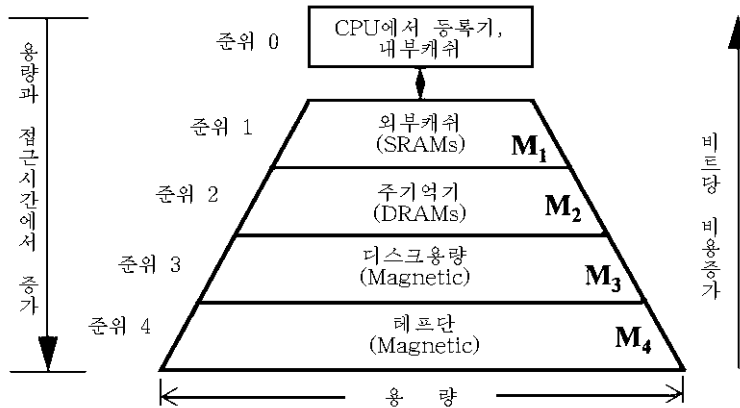



그림 5-1. 대규모콤퓨터체계를 위한 4-준위계층기억기

낮은 준위에 있는 기억기장치일수록 접근시간이 더 빠르고 byte당 비용이 더 비싸며 전송대역너비는 더 높다. 그것은 더 높은 기억기준위의 파라메터들과 비교되는 더 작은 전송단을 리용한다. 다른 말로 말하면 *n*-준위계층기억기에서 다음과 같은 부등식이 성립한다.

$$t_{i-1} < t_i,\ s_{i-1} < s_i,\ c_{i-1} > c_i,\ b_{i-1} > b_i,\ x_{i-1} < x_i \qquad (5.1)$$

현대 처리기는 대표적으로 32개 옹근수등록기들과 32개 류동소수점등록기를 가지고 있다. 이것들을 **일반목적등록기**(GPRs) 혹은 **등록기파일**이라고 부른다. 콤파일시에 가상기억기배정이 진행된다. 그것들은 실행시에 물리적등록기들로 넘어 간다.

등록기들과 소편내장(on-chip)캐쉬들은 CPU소편의 부분이다.

이 기억기모형에서 소편외장외부캐쉬는 준위 1에서 고찰된다. 주기억기는 준위 2에 있으며 하드디스크는 준위 3에, 테프단은 준위 4에 있다(그림 5-1).

다중차수캐쉬들 캐쉬 그 자체는 7개 준위의 부분계층성을 가질수 있다. 흔히 처리기에 의해 1주기동안에 접근되는 소편내장캐쉬는 준위 1에서 고찰된다.

소편외장(off-chip)캐쉬를 **2차캐쉬**라고 부른다.

해석을 간단히 하기 위하여 전체 캐쉬부분계층성을 하나의 준위로 하는 효과적인 접근시간만을 고찰한다.

소편외장캐쉬들은 흔히 정적우연접근기억기(SRAM) 적접회로소편들로 제작된다.

대부분의 소편내장 I-캐쉬와 D-캐쉬는 32KB에서 결합용량으로서 128KB까지 가진다. 소편외장캐쉬는 128KB에서 64MB로 변할수 있는데 처리기형태에 따라 설계된 외부캐쉬조종기에 의존한다. 기억기관리단(MMU)은 자료 CPU소편의 부분으로서 제작된다. MMU는 프로그람작성자에게 명백하다. 대부분의 캐쉬들은 일부를 제외하고 사용자들에게 명백하다.

주기억기 주기억기는 흔히 작은 기억기판에서 묶음화된 다중동적우연 접근기억기(DRAM)소편으로 제작되였다. SIMMS(Small in−line memory module)이라고 부른다. 1997년에 DRAMs은 소편당 4, 8, 16, 32 그리고 64Mbs에서 리용할수 있다. 128Mb에서 1Gb까지의 DRAMs이 곧 출현한다.

전형적인 주기억기용량은 현재의 PCs에서는 8에서 64MB범위에 있으며 워크스테이션에서는 32로부터 128MB, SMP봉사기에서는 128MB에서 4GB범위에 있다. 이 용량들은 앞으로 계속 늘어 날것이다.

주기억기는 흔히 고속관흐름화된 접근 혹은 아주 많은 기억기단어들을 동시에 병렬접근하기 위한 엇끼우기형기억기모듈(interleaved memory modules)로 조직된다.

기억기대역너비는 기억기모선 혹은 기억기절환을 통하여 전송할수 있는 단어들의 수 혹은 백만 byte의 수로 정의된다. 엇끼우기형기억기모듈은 또한 보다 높은 기억기대역너비뿐아니라 기억기고장을 매 기억기뱅크에서 분리시키는 높은 허용성을 제공하는 기억기뱅크로 함께 묶어 진다.

주기억기는 MMU와 동작체계가 협동하여 관리된다.

물리적기억기와 대규모디스크공간사이의 유효한 기억기관리는 주요한 OS기능의 하나이다.

만일 물리적인 SIMMs가 각이한 CPU판에 분산되면 기억기판 혹은 CPU/기억기판을 추가하여 주기억기를 확장시킬수 있다. 주기억기는 각이한 준위에서 각이한 기억기속도기술을 리용하는 다중준위에 의해 조직될수 있다.

목적은 전체 기억기비용을 줄이는것이다.

디스크구동 디스크구동과 테프단은 사용자개입을 제한시킨 OS에 의해 조종된다. 디스크기억은 직결기억기의 가장 높은 준위로 본다. 그것은 사용자프로그람과 자료모임은 물론 OS와 콤파일러와 같은 체계프로그람들을 보관한다.

자기디스크는 두가지 형태가 있는데 플로피디스크, 하드디스크이다. 플로피디스크는 흔히 1.44MB용량을 가지며 유연성구동기에 의해 이동할수 있다. 하드디스크는 지난 15년동안에 직경이 14로부터 1.8in으로 축소하였다.

이 디스크의 형식화된 자료용량은 IBM 3090콤퓨터에서 1.8in Integral 1820디스크구동의 20MB에서 10.88in 디스크구동의 22.7GB범위에 있다.

디스크접근시간은 10에서 25ms범위에 있는데 기계적회전으로 아주 느리다. 자료전송률은 1.9에서 4.2MB/s범위에 있다. 자기디스크는 전력중단으로 변화되지 않지만 기계적고장이 일어 나기 쉽다.

그러므로 디스크에 기억된 정보는 다음준위의 기억기인 테프단에 돌려 보내야 한다.

테프단 자기테프단은 여유기억기로서 비직결기억기를 쓴다.

테프단접근시간은 초단위정도로 느리다. 그러나 용량을 100GB 지어는 TB로 올릴수 있다.

테프단은 현재와 과거의 사용자프로그람들, 실행결과들 그리고 처리된 파일들의 복사를 보관한다.

테프에 정보파일들을 기억하면 목적이 달성된것으로 본다. 테프에 기억된 정보파일들은 직결식이 아니다.

그것들을 복귀하기 위하여 콤퓨터동작기들이 개입되여야 한다.

표 5-1은 주기억기, 디스크구동장치와 테프단의 물리적인 특징들을 비교한다. 단위용량은 단위구역에 기억가능한 정보를 표시한다. 체계용량은 높은급워크스테이숀콤퓨터의 용량을 표시한다. 현재값은 1997워크스테이션 /Server판매자들에 의해 광고된 수이다.

표 5-1 높은급워크스테이션의 기억기, 디스크구동장치와 테프단

파라메터	값	기억기	디스크구동	테프단
용 량	단가	64 Mb/die	0.2 Mb/mm^2	0.1-0.4 Mb/mm^2
	체계	32-128 MB	2-4 GB	8-16 GB
	개선률	2X per 2 yr	25-60% per yr	10X per 10-21 yr
접근시간	현재값	60 ns	10-25 ms	seconds
	개선률	3X per 12 yr	2X per 10 yr	N/A
대역너비	현재값	500-1000 MB/s	1-5 MB/s	0.5-4 MB/s
	개선률	N/A	2X per 5 yr	2X to 6X per 10 yr

5. 1. 2. 계층기억기의 성질

아래에서 서술되는 기본적인 성질들은 유효한 계층기억기를 설계하거나 리용하기 위한 포함, 일관성 그리고 국부성이다.

계층기억기의 설계법칙은 본질적으로 이 성질들로부터 유도된다. 이 성질들은 린접한 기억기준위들사이의 관계를 설정한다. 그것들은 또한 계층기억기의 동작을 규정한다.

포함성질 포함성질은 n개 기억기준위들사이의 모임포함관계를 다음과 같이 표시한다.

$$M_1 \subset M_2 \subset M_3 \subset \cdots \subset M_n \tag{5.2}$$

두개의 린접한 준위들사이의 모임포함관계는 준위 Mi+1에 기억된 모든 정보대상들(단어, 캐쉬행 혹은 페지)은 M_{i+1} (모든 $i=1, 2, \cdots, n$) 에 기억된 정보대상들의 부분모임을 이룬다는것을 의미한다. 초기에 M_n은 프로그람을 실행하는데 필요되는 모든것을 포함한다. 실행주기동안에 M_n의 부분모임들이 다음의 낮은 준위 M_{n-1}로 복사된다. 류사하게 M_{n-1}의 부분모임은 M_{n-2}에 복사되며 요구되는 명령과 자료가 기억기 M_1로 이동할 때

까지 계속하여 처리기에 의한 실행을 준비한다.

M_i에서 정보를 찾을수 있다면 같은 정보의 복사를 보다 높은 준위 $M_{i+1}, M_{i+2}, \cdots, M_n$에서 찾아 볼수 있다. 그러나 M_{i+1}에 기억된 단어는 M_i에서 찾을수 없다.

M_i에서 단어를 찾지 못하면 모든 낮은 준위 $M_{i-1}, M_{i-2}, \cdots, M_1$에서도 찾지 못한다. 가장 높은 준위는 여유기억이며 여기서 모든것을 찾을수 있다.

정보복사에 관하여 M_i는 M_{i+1}의 참부분모임이다.

자료전송단 이것은 린접한 기억기준위사이의 정보전송단을 가리킨다. 기억기단어(4b는 32b기계를 위한것이고 8B는 64b기계를 위한것)는 CPU등록기들과 내부소편내장캐쉬사이의 전송단이며 캐쉬행은 내부캐쉬와 M_1에서의 외부캐쉬사이의 전송단이다. 최근의 일부 기계들은 캐쉬행크기를 64b 혹은 128B로 확장하였다.

M_2에서 주기억기는 흔히 페지당 4KB인 고정크기페지로 나눈다. 각이한 기계는 각이한 페지크기를 선택할수 있다. 페지는 디스크와 주기억기사이의 자료전송단이다.

분산형페지들은 흔히 디스크에서 파일들의 토막으로 국부적으로 묶어 진다. 그러므로 토막은 테프단과 디스크사이의 전송단이다. 단어는 캐쉬행의 부분이고 캐쉬행은 페지의 부분이며 페지는 토막의 부분이라는것을 명백히 해야 한다.

일관성 일관성은 같은 정보항목들의 복사들이 각이한 기억기수준에서 일치할것을 요구한다.

단어 혹은 캐쉬행이 낮은 수준에서 수정되면 그 단어 혹은 캐쉬행의 복사들은 높은 기억기준위에서 즉시에 갱신되여야 한다. 현재 쓰인 명령/자료항목들은 높은 준위의 기억기들로부터 캐쉬에로 빨리 축적된다.

따라서 일관성문제는 M_1의 캐쉬에서 기억기 M_4로 모든 방식을 확장한다. 련속적인 준위에서 기억기일관성을 유지하는것은 중요한 문제이며 종종 외부모선주기를 요구하거나 기억기지연시간을 연장한다. 일반적으로 계층기억기에서 일관성을 유지하는데 두가지 방법이 있다. 첫번째 방법을 련속쓰기(write−through,WT)라고 부르는데 단어가 $M_i(i=1,2,\cdots,n-1)$에서 수정되면 M_{i+1}에서 즉시적인 갱신을 요구한다. 두번째 방법은 후쓰기(write−back, WB)인데 수정된 정보항목을 포함하는 M_i의 기억기단이 다른 새로운 정보항목으로 바꾸어 질 때까지 M_{i+1}에서의 갱신을 지연시킨다. 따라서 기억기설계는 WT와 WB로 나눈다.

실천적으로 WT는 내부캐쉬(I−캐쉬 혹은 D−캐쉬)를 갱신하는데 쓴다. WB는 M_1기억기인 외부2차캐쉬를 갱신하는데 쓴다. 잘 알려 진 기억기교체방법은 LRU(가장 최근에 쓰인것, LFU(최소리용빈도), FIFO(선입선출)와 우연알고리듬 등을 포함한다.

참조국부성 계층기억기는 참조국부성으로 알려 진 프로그람동작에 기초하여 개발된다. 많은 참조들은 명령 혹은 자료접근을 위한 CPU에 의해 발생된다.

이 접근들을 시간, 공간, 순서화의 령역에서 아래와 같이 정의한다.

(1) **시간국부성** 최근에 참조된 항목들(명령 혹은 자료)은 앞으로 다시 참조될수 있다. 이것은 흔히 반복고리, 처리탄창, 시간변수 혹은 부분프로그람과 같은 전용프로그람구조체에 기인된다. 일단 고리(순환)에 들어 가거나 부분프로그람을 호출하면 작은 코드토막이 수많이 반복적으로 참조된다. 따라서 시간국부성은 가장 최근에 쓰인 코드와 자료토막주위로 클라스터화된다.
(2) **공간국부성** 이것은 쏘프트웨어처리가 주소가 서로 가까이에 있는 정보항목들에 접근하는 경향성을 가리킨다. 실례로 표나 배렬에 있는 동작들은 주소공간의 일정한 클라스터화된 구역에 대한 접근을 포함한다. 루틴과 마크로와 같은 프로그람토막들은 기억기공간의 같은 린접에 기억되는 경향이 있다.
(3) **순차국부성** 대표적인 프로그람에서 명령실행은 분기명령이 비순차실행을 발생하지 않는 한 프로그람순서라고 부르는 순차에 따른다. 비순차실행에 대한 순차실행률은 보통 프로그람에서 대체로 5에서부터 1까지이다. 또한 대규모자료렬의 접근도 행렬원소의 주요행(row-major)접근과 같은 순차를 따른다.

기억기설계실현 프로그람수행에서 순차국부성은 모든 형태의 국부성과 관계한다. 왜냐하면 순차적으로 코드화된 명령들과 배렬원소들이 린접하는 기억기장소에 기억되고 거의 같은 시간에 참조되기때문이다. 국부성의 매 형태는 계층기억기설계에 영향을 준다.

- 시간국부성은 기억기설계를 위한 LRU교체알고리듬리용에 맞게 동작한다. 그것은 후에 소개되는 작업모임개념과 관계된다.
- 공간국부성은 린접한 기억기준위들사이의 단위자료전송크기를 결정하는데 도움을 준다. 시간장소 역시 련속준위에서 기억기크기를 결정하는데 도움을 준다.
- 순차국부성은 최량일정짜기를 위한 grain크기를 결정하는데 영향을 준다. 미리꺼내기기술은 이 국부성성질로부터 영향을 받는다.

5. 1. 3. 기억기용량의 계획작성

계층기억기성능은 계층의 어느 준위에 대한 유효접근시간 T_{eff}에 의해 결정된다.
그것은 련속준위에서 적중률(hit ratios)과 접근빈도에 의존한다.
아래에서 이 항목들을 형식적으로 정의한다. 그다음 주어 진 비용제한에서 계층기억기용량을 최량화하기 위한 모형을 제기한다.

적중률 적중률은 두개의 어떤 린접한 계층기억기준위들사이에서 정의된다. 정보항목을 M_i에서 찾을 때 그것을 적중한다고 말하고 그렇지 않으면 비적중이라고 말한다.
N-준위계층기억기를 고찰하자. 적중률 h_i는 정보항목이 M_i에서 찾게 되는 확률이다. 그것은 준위 M_{i-1}과 M_i의 특징에 대한 함수이다. 비적중률은 $1-h_i$로 정의한다.

련속준위에서 적중률은 기억기용량, 관리방법 그리고 프로그람동작의 함수이다. 계층에서 련속적인 적중률은 0과 1사이의 값을 가진 독립우연변수이다. 간단히 유도하기 위해 $h_0=0$, $h_n=1$이라고 가정한다. 이것은 CPU소편안에서 비적중후 M_1에 대한 접근을 초기화한다는것과 그리고 M_n에 대한 접근은 언제나 적중하다는 결과를 얻는다는것을 의미한다.

M_i에 대한 접근빈도는

$$f_i = (1-h_1)(1-h_2)\cdots(1-h_{i-1})\, h_i \tag{5.3}$$

로 정의한다.

이것은 모든 낮은 준위에서 $i-1$번 비적중했을 때 준위 M_i에서는 적중하는 확률이다.

국부성성질에 의해 접근빈도는 낮은 준위에서 높은 준위에로 아주 빨리 감소한다. 즉

$$f_1 \gg f_2 \gg f_3 \cdots \gg f_n \tag{5.4}$$

이것은 기억기내부준위는 외부준위보다 훨씬 자주 접근된다는것을 의미한다. 대표적으로 첫번째 준위 f_1은 95%보다 더 빠르고 나머지준위들은 크기감소순서에 따라 합해서 나머지 5%로 된다.

유효접근시간 실천적으로 M_1에서 가능한 높은 적중률을 얻는것이 좋다. 매번 비적중이 일어 나면 다음번 높은 기억준위에 접근하기 위해 벌칙을 가하여야 한다.

비적중을 주기억에서는 **페지오유** 혹은 캐쉬에서는 캐쉬행비적중이라고 불렀다. 그것은 캐쉬행과 페지는 이 준위들사이의 전송단이기때문이다.

페지오유에 대한 시간벌칙은 $t_1 < t_2 < t_3$이므로 캐쉬행에서보다 훨씬 길다. Stone(1990)은 캐쉬비적중은 캐쉬적중보다 2~4배 비용이 더 많이 들지만 페지오유는 페지적중보다 1000~10000배 비용이 더 많이 든다는것을 지적하였다.

접근빈도 $f_i,\ i=1,2,\cdots,n$을 리용하여 계층기억기의 유효접근시간을 다음과 같이 정의한다.

$$\begin{aligned} T_{\text{eff}} &= \sum_{i=1}^{n} f_i \cdot t_i \\ &= h_1 t_1 + (1-h_1)\, h_2 t_2 + (1-h_1)\,(1-h_2) h_3 t_3 + \cdots \\ &\quad + (1-h_1)(1-h_2)\cdots(1-h_{n-1}) t_n \end{aligned} \tag{5.5}$$

식 (5.5)에서 앞의 몇개 항은 전체 값의 많은 몫을 차지한다. 왜냐하면 *f1*과 *f2*이 다른 것들보다 훨씬 크기때문이다. 유효접근시간은 프로그람동작과 기억기설계선택에 의거한

다. 프로그람추적을 한후에만 적중률과 T_{eff}값을 정확히 평가할수 있다.

계층최량화 계층기억기의 전체 값은 다음과 같이 평가된다.

$$C_{total} = \sum_{i=1}^{n} c_i \cdot s_i \tag{5.6}$$

이것은 비용이 n개의 준위들에 분포되여 있다는것을 의미한다.

$c_1 > c_2 > c_3 \cdots > c_n$ 이므로 $s_1 < s_2 < s_3 \cdots < s_n$ 을 선택해야 한다. 계층기억기의 최량설계는 M_1의 t_1에 가까운 T_{eff}에 귀착되고 전체 비용은 기억기준위 M_n의 C_n에 가깝다. 실제상 이것을 얻는것은 n개준위가운데서 절충(tradeoffs)때문에 어렵다. 최량화과정은 선형계획문제로 형식화될수 있다. 전체 비용의 최고한도를 C_0으로 준다.

다른말로 말하면 다음과 같은 제한밑에서 T_{eff}를 최소화하려고 한다.

$$s_i > 0,\ t_i > 0,\quad i = 1, 2, \cdots, n$$
$$C_{total} = \sum_{i=1}^{n} c_i \cdot s_i < C_0 \tag{5.7}$$

표 5-2에서 보여 준바와 같이 매 준위 M_i에서 단위비용 C_i와 용량 S_i는 속도요구 혹은 요구되는 접근시간 t_i에 관계된다.

그러므로 우의 최량화는 t_i, c_i, s_i, f_i 혹은 h_i(모든 준위 $i = 1, 2, \cdots, n$)가운데서 절충을 포함한다.

다음의 실례는 선형계획문제를 풀어 어떻게 계층기억기를 설계하는가를 보여 준다.

실례 5.1. 계층기억기설계

기억기에 대한 다음의 특성을 가진 3-준위계층기억기설계를 고찰하자. 설계목적은 캐쉬적중률 h_1=0.98, 주기억기적중률 h_2=0.9인 유효기억기접근시간 T_{eff}=10.04μs를 얻는것이다. 계층기억기의 전체 비용은 웃한계가 15000$이다. 계층기억기비용은 $c = c_1 s_1 + c_2 s_2 + c_3 s_3 < 15{,}000$로 계산된다.

표 5-2 기억기부분체계실례에서 파라메터들

기억기준위	접근시간	용량	비용/KB
캐쉬	T_1=25ns	S_1=512KB	C_1=$1.25
주기억기	T_2=미지	S_2=32MB	C_2=$0.2
디스크배렬	T_3=4ns	S_3=미지	C_3=$0.0002

디스크최대용량은 최대비용을 초과함이 없이 s_3=39.8GB로 얻어 진다. 다음으로 주기억기를 제작하기 위한 RAMs의 접근시간 t_2을 선택하자.

유효기억기접근시간은 다음과 같다.

$$T_{\text{eff}} = h_1 t_1 + (1-h_1)h_2 t_2 + (1-h_1)(1-h_2)h_3 t_3 < 10.04\mu\text{s} \qquad (5.8)$$

우의 값들을 대입하면 $10.04 \times 10^{-6} = 0.98 \times 25 \times 10^{-9} + 0.02 \times 0.9 \times t_2 + 0.02 \times 0.1 \times 1 \times 4 \times 10^{-3}$. 따라서 디스크접근시간은 t_2=903ns로 선택된다.

같은 최대비용제한에서 디스크용량을 감소하는 대신에 주기억기를 64MB로 두배 하려고 한다고 하자.

이 변화는 캐쉬적중률에 영향을 주지 않는다. 그러나 알맞는 교체알고리듬을 리용하면 주기억기에서의 적중률을 증가시킬수 있다. 그러면 전체 계층의 총체적인 유효접근시간은 감소될것이다.

5. 2. 캐쉬일관성규약

이 절에서는 캐쉬일관성규약을 고찰한다. 일관성규약의 선택은 프로그람실행의 정확성은 물론 공유기억기성능에 영향을 준다. 일관성문제는 공유기억기에서 주소화된 같은 캐쉬행을 가진 분산캐쉬복사의 불일치성을 가리킨다.

5. 2. 1. 캐쉬일관성문제

기억기체계에서 캐쉬행 *X*의 어떤 캐쉬복사에 대한(읽기 혹은 쓰기를 위한) 접근이 언제나 다음과 같은 캐쉬결과를 가지고 되돌아 오면 기억기부분체계는 캐쉬행준위에서 일관된다고 본다([492], 656p). 즉

- 처리기 P에 의한 *X*쓰기(다른 처리기에 의한 *X*쓰기는 없다.)다음의 읽기는 P에 대한 씌여 진 값을 복귀한다.
- 다른 처리기 Q에 의한 *X*쓰기다음에 처리기 P에 의한 읽기는 읽기와 쓰기가 충분히 갈라 지고 그사이에 다른 처리기들에 의한 *X*쓰기가 없으면 Q가 쓴 값을 복귀한다.
- 여러가지 처리기들에 의하여 같은 캐쉬행 *X*에 대한 쓰기는 기억기사건들의 같은 순서를 제시하기 위해 언제나 련속적으로 진행된다.

무일관성원인 캐쉬무일관성은 아래에서 서술한바와 같이 3가지 가능한 원인을 가진다.

(1) 기억기에서 같은 캐쉬행의 캐쉬복사에 대한 여러가지 처리기의 비동기적쓰기
(2) 다중처리기가운데서 서로 감시 없는 프로쎄스이동
(3) 캐쉬복사의 소유권을 무시하는 I/O동작들

이 원인들을 캐쉬일관성조종이 없는 공유기억기2중처리기체계에서 간단한 실례를 들어 설명하자.

매 처리기는 전용캐쉬를 가진다. 우리는 WT와 WB캐쉬들에 대한 영향을 각각 고찰한다.

WT캐쉬의 경우에 기억기캐쉬행은 캐쉬에서 가장 최근의 변경과 언제나 일치한다. WT캐쉬는 매 쓰기다음의 기억기를 갱신하는데 더 많은 모선주기를 소비한다.

WB캐쉬의 경우 기억기복사는 교체가 일어 날 때까지 갱신되지 않을것이다. 그러므로 기억기에서 캐쉬행은 캐쉬에서 쓰기적중다음에 즉시에 캐쉬복사와 차이날수 있다.

기억기후쓰기는 그렇게 하는것이 절대적으로 필요할 때까지 지연된 련속쓰기이다.

그러므로 WB캐쉬는 기억기모선에서 실행하는데 보다 경제적이다.

실례 5.2. 공유자료쓰기에 의한 무일관성캐쉬들

캐쉬쓰기전과 후에 캐쉬상태에서의 변화를 그림 5-2에 제시한다.

갱신전에 두개 처리기에서 두개 캐쉬복사(X로 표시됨)는 캐쉬행 X와 공유기억에서 일치한다고 가정한다.

WT캐쉬를 리용하여 처리기 P_1가 그것의 캐쉬복사를 x'로 변화시킨후 기억기복사도 x'로 변화되여 다른 캐쉬복사와 차이나게 한다.

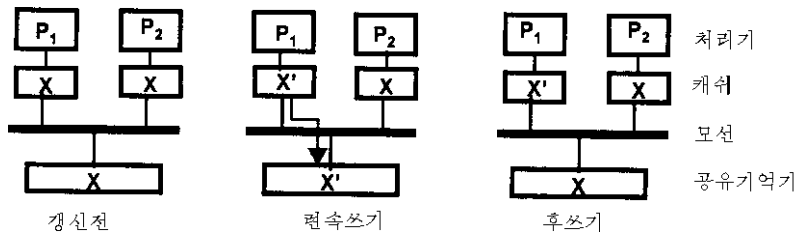



그림 5-2. 쓰기가능한 자료공유에 의한 캐쉬무일관성

WB캐쉬의 경우에 캐쉬 1에서의 변화는 캐쉬 2 혹은 기억기복사에서 변화를 일으키지 않으며 그것들은 캐쉬 1에서의 가장 최근의 쓰기와 차이난다.

이 실례는 명백히 캐쉬들을 무효로 하거나 매 캐쉬쓰기동작후에 그것들을 즉시에 갱신할 필요를 보여 준다. 왜냐하면 쓰기가능한 자료는 흔히 다중극소형처리기체계에서 공유되기때문이다.

실례 5.3. 프로쎄스이동에 의한 무일관성캐쉬들

이제 P_1가 기억기에서 기준 x의 캐쉬복사를 가진다고 하자.

P_1에서 P_2에로 프로쎄스이동후 P_2은 같은 기준에 새로운 내용 y를 쓴다.

이것은 그림 5-3 중간에서 보여 준바와 같이 WT캐쉬들을 리용하는데서 캐쉬 1과 기억기사이의 불일치성을 초래할것이다.

다른 한편 WB캐쉬들이 오른쪽에서 보여 준바와 같이 리용될 때 이동이 후쓰기동작앞에서 진행된다면 낡은 값 x는 읽기동작을 통하여 캐쉬 2로 이동될수 있다.

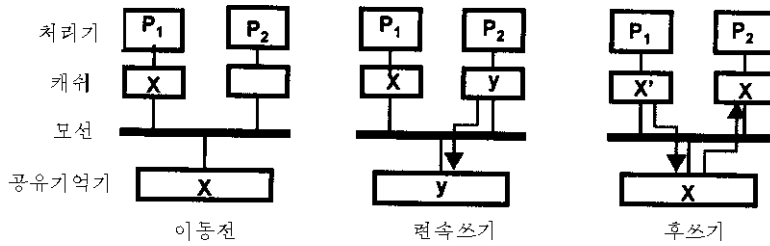



그림 5-3. 처리기이동에 의한 캐쉬무일관성

두가지 경우에 프로쎄스이동은 캐쉬무일관성을 일으킨다.

실례 5.4. 캐쉬를 무시하는 I/O동작들에 의한 무일관성

입구/출구동작들은 그림 5-4에서 보여 주는바와 같이 무일관성을 일으키는 캐쉬를 무시할수 있다. 여기서 I/O통은 WT캐쉬를 리용하는 입구장치와 WB캐쉬를 위한 출구장치이다. 초기에 두 캐쉬복사들은 기억기에서 캐쉬행과 일치한다.

입구장치는 새로운 내용 x′ 를 가진 기억기기준을 적재할수 있는데 두 캐쉬복사의 어느 하나와도 차이가 나지 않게 한다.

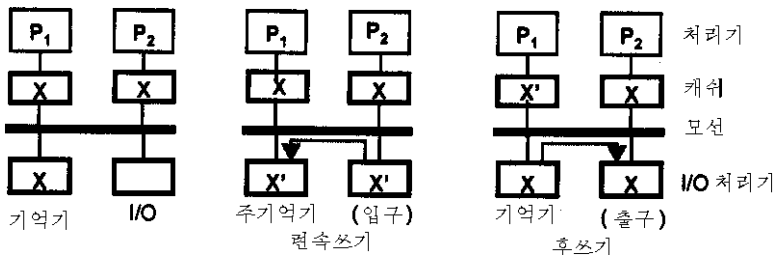



그림 5-4. 캐쉬를 무시하는 I/O동작에 의한 무일관성

WB캐쉬들을 가진 출구인 경우에 캐쉬 1은 수정될수 있지만 낡은 값 x'는 출구장치로 읽는다.

넘기기되고 있는 통보문은 I/O동작들의 변화에 대해 캐쉬조종기가 주의를 돌리도록 하는것이다.

5. 2. 2. Snoopy 일관성규약

우의 실례들은 극소형처리기환경에서 공유자료, 프로쎄스이동, I/O동작들의 쓰기에 어떤 일관성규약을 적용할 필요를 암시한다.

공동모선에 련결된 각이한 처리기들에 의해 접근된 전용캐쉬들을 리용하는데서 두 클라스의 일관성규약은 쓰기-무효규약과 쓰기-갱신규약이다.

본질적으로 쓰기-무효규약은 국부캐쉬복사가 갱신될 때 다른 모든 캐쉬복사들을 무효로 한다.

쓰기-갱신규약은 같은 기준주소를 가진 다른 모든 캐쉬복사를 갱신하기 위해 새로운

캐쉬복사를 방송한다.

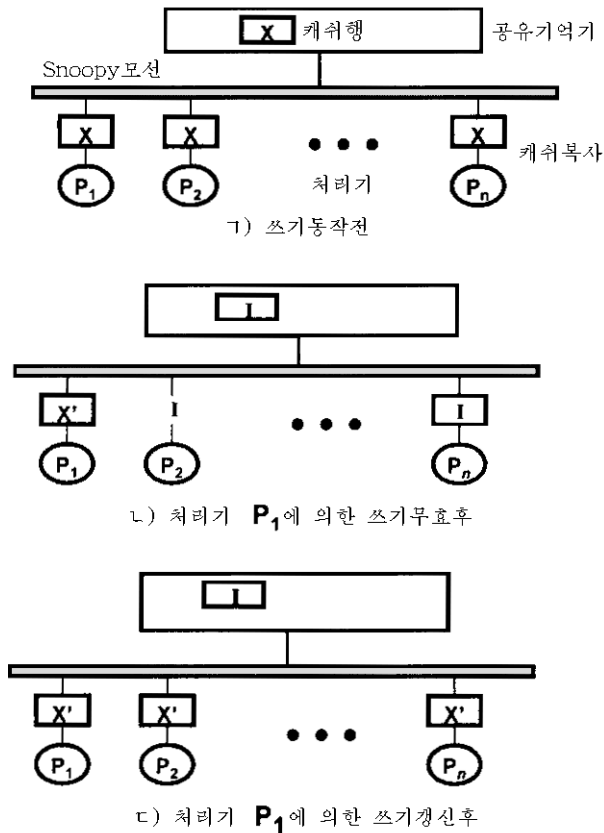



그림 5-5. 후쓰기캐쉬들을 리용하는데서 쓰기-무효와 쓰기-갱신 Snoopy 규약들
(x:원래 캐쉬복사, x':수정된 캐쉬행, I:무효된 캐쉬행)

이 캐쉬일관성규약은 Snoopy모선들의 리용으로 실행된다.

Snoopy규약들은 모선 혹은 고리에 의해 보장된것과 같은 방송(broadcast)기구를 요구한다.

기본방법은 처리기와 기억기모쥴사이의 모선을 통과하는 캐쉬사건들을 계속 감시하기 위한 모선을 설계하는것이다. 이런 의미에서 Snoopy**일관성규약**이라고 부른다.

WB캐쉬의 경우에 대한 두개의 Snoopy규약들을 그림 5-5에 제시한다.

WT캐쉬들을 리용할 때 단지 차이는 매 쓰기동작다음 기억기기준을 갱신하는 시간측정에 있다.

쓰기-무효규약 그림 5-5 ㄱ)에서 캐쉬행 x에 대한 3개의 캐쉬복사들의 존재를 보여준다.

처리기가 x에 쓰려고 할 때 먼저 x에 접근하기 위한 배타적인 권한이 있어야 한다. 그다음 새로운 내용 x'로 캐쉬행을 쓰고 다른 처리기에서 x의 모든 캐쉬복사를 무효로

한다.

WB캐쉬에 대하여 기억기준은 그림 5-5 ㄴ)에서 보여 준바와 같이 역시 무효로 된다.

WT캐쉬가 리용되면 기억기기준은 변화된 값 x'로 즉시에 채워 진다.

무효된 캐쉬복사들은 그림 5-5 ㄴ)에서 I로 표시한다. 어떤 다른 처리기가 변경된 캐쉬행 x'에 접근하려 할 때 WT가 리용되면 처리기 P1의 캐쉬복사로부터 캐쉬비적중을 찾고 새로운 값 X'를 얻을것이다.

WB캐쉬가 쓰인 경우에 새로운 값을 변경된 캐쉬로부터 다시 찾을것이며 기억기기준은 동시에 채워 진다(갱신된다.).

쓰기갱신규약 이 경우에 처리기는 캐쉬행 x에 쓰고 모든 처리기들에서 x의 모든 캐쉬복사를 갱신하여야 한다(즉 새로운 값 x'를 x의 모든 캐쉬복사에 방송하여야 한다). 어떤 다른 처리기가 새로운 캐쉬복사 x'에 접근하려고 할 때 그림 5-5 ㄷ)에서 보여 준바와 같이 그것의 국부캐쉬에서 캐쉬적중(hit)을 만나고 새로운 값을 찾을것이다.

기억기기준은 그림 5-5 ㄷ)에서 보여 준바와 같이 WB캐쉬리용으로 갱신될것이다. 그러나 기억기 역시 실행선택에 따라 즉시 갱신될수 있다.

명백히 쓰기-갱신규약은 언제나 일관성정도를 더 높인다. 그러나 쓰기갱신은 비용이 많이 드는 동작이다. 즉 쓰기갱신은 기억기에서 캐쉬행은 물론 모든 캐쉬들을 갱신하기 위한 모선주기들을 더 많이 소비할수 있다.

그러므로 대부분의 다중처리기설계자들은 WB캐쉬를 리용하는 쓰기-무효캐쉬일반성규약을 선택한다.

다음절에서 MESI무효규약을 고찰하게 될것이다.

기억기모선에서 Snoopy규약외에 캐쉬일관성은 주로 NUMA 혹은 DSM모형을 가진 망으로 련결된 다중처리기들에서 등록부에 기초한 규약들에 의해 만족될수 있다.

5. 2. 3. MESI Snoopy 규약

MESI는 쓰기-무효 snoopy규약이다. 그것은 캐쉬행상태의 추적을 보존하며 모든 읽기 혹은 쓰기, 캐쉬적중 혹은 캐쉬비적중 그리고 모선에서 검측된 snoopy사건들을 고찰한다.

여기서는 그림 5-6에서 보여 준 Pentium에 기초한 다중처리기에서 실행되는 MESI규약을 례로 하여 고찰한다.

Pentium MESI는 외부신호에 의해 조종되는 WB와 WT캐쉬사건들을 지원한다. 규약은 비-쓰기-배당방법을 선택하는데 쓰기비적중의 기억기로부터 캐쉬행충족이 없다는것을 의미한다.

MESI규약은 4개 상태이행도표인 그림 5-7에서 보여 준다.

모든 기억기사건들과 Snoopy신호들은 상태들사이의 가지(호)에 대응한다. 도표에서

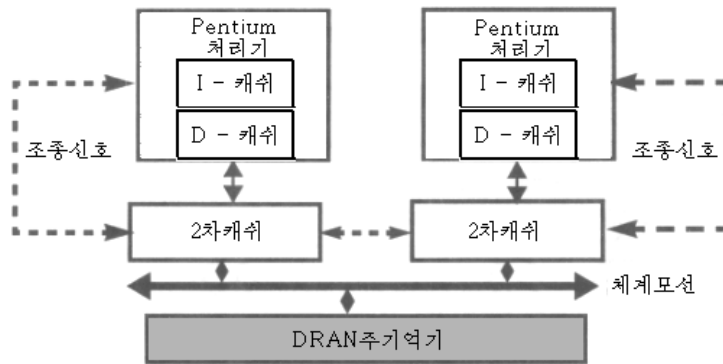



그림 5-6. 2 차캐쉬를 가진 2 중 Pentium 다중처리기

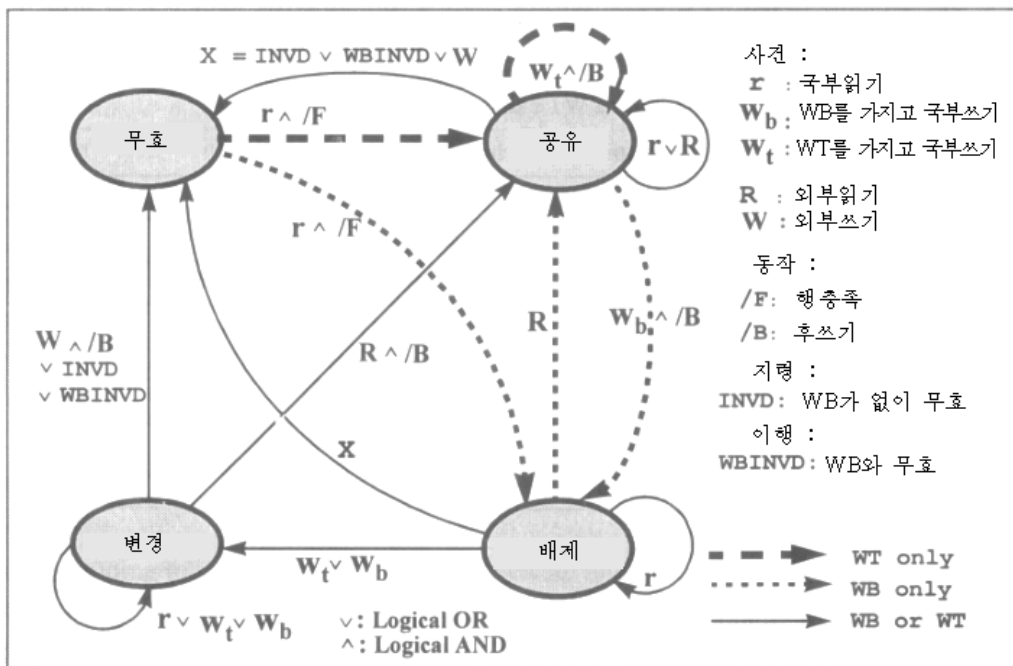



그림 5-7. Pentium에 기초한 다중처리기에서 자료캐쉬를 위한 MESI 규약의 상태이행도표

3개의 가능한 이행가지를 볼수 있다.

굵은 점선은 WT규약에만 적용한다. 얇은 점선은 WB캐쉬사건에만 적용한다. 실선은 WB 혹은 WT규약에 대한것이다. 자료캐쉬에서 매 기준은 다음과 같은 4개의 가능한 상태중의 어느 하나이다.

- **변경(Modified, M상태)** 캐쉬행이 캐쉬에서 국부처리기에 의해 쓰기적중으로 갱신되였다.

- **배제 (Exclusive, E상태)** 캐쉬가 유효이고 어떤 다른 캐쉬에서 유효가 아니다. 더우기 기억기기준은 아직 갱신되지 않았다.
- **공유(Shared, S상태)** 기준이 유효이지만 하나 혹은 그이상의 원격캐쉬 혹은 기억기의 캐쉬행에서 유효가 될수도 있다.
- **무효(Invalid, I상태)** 이것은 재설정다음의 초기캐쉬상태이거나 캐쉬행이 같은 주소를 가진 다른 캐쉬로 인한 쓰기적중에 의해 무효되였다.

MESI라는 이름은 우의 4개 상태의 첫 글자로부터 만든것이다. 한 상태에서 다른 상태에로의 이행은 국부처리기읽기, 쓰기 혹은 다른 주모선으로 초기화된 외부 Snoopy동작에 의해 일어 난다.

무효명령(INVD 혹은 WBINVD)리용은 프로그람작성자들에 의해 초기화된다. 이것은 재기동 혹은 문맥절환에서 체계프로그람작성자들이 캐쉬를 려기시키기 위한 선택권을 제공한다.

사용자프로그람은 보통 이 명령들을 리용하지 않도록 한다.

여러가지 각이한 Snoopy규약들을 명백히 하기 위하여 표 5-3에 그림 5-7의 모든 MESI상태이행에 대응하는 캐쉬사건들과 snoopy사건들을 요약하여 제시하였다.

표 5-3 캐쉬사건들과 snoopy모선사건들

사건기호	사건, 동작, 결론의 해석
M	변경:오물복사;읽기 혹은 쓰기적중; 무효가 되면 요구되는 후쓰기
E	배제:단독복사;읽기적중, 국부쓰기다음에 M상태로 변환
S	공유:청소복사;읽기적중,다른 캐쉬를 무효시키기 위한 쓰기;2개이상의 복사가 WB캐쉬사건을 위한 체계에 존재한다.
I	무효:캐쉬되지 않은, 읽기 혹은 쓰기 비적중
R	국부읽기동작
W_t	WT캐쉬기준에 국부쓰기
W_b	WB캐쉬기준에 국부쓰기
R	읽기에서 외부 snoopy적중
W	쓰기에서 외부 snoopy적중
/F	주기억기로부터 이 캐쉬기준을 충족한다.
/B	후쓰기:모든 캐쉬들을 무효하고 주기억기를 갱신
INVD	자기 캐쉬를 려기시키는 체계지령
WBINVD	후쓰기다음에 자기캐쉬를 려기시키는 체계지령
X	INVD 혹은 WBINDB 혹은 W의 사건결합

그림 5-7은 WT와 WB캐쉬사건들의 지원이 있을 때 두개 이행도표의 결합을 보여 주는것인데 하나는 WT캐쉬사건에 대한것이고 다른것은 WB캐쉬사건에 대한것이다.

그것들은 일부 상태와 이행가지를 공유한다.

간단하고 명백히 하기 위하여 이 두가지 경우들을 각각 서술한다.

련속쓰기규약 WT캐쉬행에 대하여 초기상태는 I이고 다른 가능한 상태는 S만이다. 비적중이 있을 때 순수한 캐쉬행은 주기억기 혹은 다른 캐쉬에서 나오고 캐쉬행은 공유상태에서 끝난다. 모든 이행들은 이 두 상태들에서 닫긴다. 그림 5-7의 웃절반에 있는 S와 I 그리고 굵은 점선과 실선이행을 가지고 이루어 진 두 상태부분그라프를 WT캐쉬사건에 대한 **SI규약**이라고 부른다.

캐쉬행은 읽기비적중다음에 캐쉬로부터 채워 질것이다.

또한 다른 처리기들은 어떤 처리기가 이 기준을 다시 쓸 때까지 이 기준을 읽고 캐쉬할수 있다.

이 쓰기동작에 따라 쓰기처리기는 먼저 모든 다른 캐쉬복사를 무효로 한 다음 새로운 자료들을 자기의 캐쉬와 기억기에로 련속쓰기한다.

쓰기상태는 S상태에 머물러 있고 다른것들은 무효상태 I로 간다.

후쓰기캐쉬들 WB캐쉬에서 기준은 E상태의 캐쉬행으로 넘어 가는데 이것은 체계에서 유일한 캐쉬복사라는것을 의미한다. 하나 혹은 그이상의 처리기들이 이 블로크를 읽고 캐쉬한 다음 그것의 모든 상태들은 S로 변화하는데 이것은 두개 혹은 그이상의 캐쉬복사가 존재한다는것을 의미한다.

WT와 류사하게 국부쓰기가 S상태에서 일어 날 때 쓰기동작은 모든 다른 상태를 무효로 하고 자기 상태를 M으로 변화시킨다.

M상태는 다른 처리기가 읽기 혹은 쓰기를 요구할 때 후쓰기필요가 있는 캐쉬행을 가리킨다.

그러나 그것이 일어 나기전에 국부처리기는 다른 캐쉬들에 통보함이 없이 기준을 임의로 읽기 혹은 쓰기할수 있다.

외부 Snoopy사건들에 대해서는 아주 강하다.

모든 원격쓰기는 다른 캐쉬들을 I상태(필요하면 먼저 후쓰기)로 변화시킨다. 모든 원격읽기는 S상태에서 적중한다.

무효 I상태에서 밖으로 나가는 쓰기가지는 Pentium의 비쓰기-배당법때문에 없다.

WB캐쉬를 위한 MSI규약 MESI규약을 리해하기 위하여 그림 5-7을 더 세밀히 관찰하자.

E상태를 단일S마디를 가진 S상태와 결합한다면 WB캐쉬에서의 사용자만을 위한 3상태 MSI규약을 얻을수 있다.

MESI상태도표로부터 MSI부분상태도표를 뽑으면 다음과 같은 식이 얻어 진다.

$$I \xrightarrow{r} E, \qquad (5.9)$$

캐쉬준위의 단독복사상태에 대응하여

$$E \xrightarrow{R} S\ , \qquad (5.10)$$

두개 혹은 그이상의 캐쉬복사가 존재한다는것을 지적하면

$$S \xrightarrow{w_b} M\ . \qquad (5.11)$$

WB동작에 대응하는것은 첫번째 쓰기에서만 출현하고 그다음의 쓰기들에서는 출현하지 않는다.

직관적으로 식 (5.11)은 다음읽기전에 하나 혹은 그이상의 처리기들에 의해 같은 준위에 다중쓰기가 있을 때 한번 쓴다는것을 의미한다.

WT캐쉬들을 위한 SI규약 3-상태 MSI규약을 WT캐쉬들을 위한 2-상태 SI규약으로 간단화할수 있다.

지어 1-상태규약도 가능하므로 캐쉬가 전혀 없는것도 있다. Snoopy규약에 대하여 일관성규약의 상태수는 공유정도에 따라 캐쉬준위를 분류하는 용량을 표현한다.

Snoopy규약들의 개괄 여기서는 여러가지 형태의 캐쉬들과 여러가지 준위-채우기 그리고 쓰기-배당방법에 대한 세가지 Snoopy쓰기-무효규약들을 고찰하였다.

일반적으로 보다 많은 상태들이 일관성규약을 사용할수록 캐쉬사건들을 보다 가능한 이행들로 분리할수 있다. 물론 비용이 높을수록 실행효률성에 의해 보다 복잡한 규약들에 귀착될것이다.

WT방법은 1차자료캐쉬에서 언제나 쓰이고 WB는 언제나 2차캐쉬에서 기억기에로 쓰인다는 가정하에서 그림 5-7에 렬거된 MESI규약을 조금 변경시켜 2차캐쉬에 적용할수 있다.

5. 3. 공유기억기일치성

공유기억기의 읽기와 쓰기는 기억기사건들이 정확히 순서화되지 않는다면 일치성문제에 맞다들린다. 이 문제를 처리하기 위하여 유효기억기모형이 공유기억기다중처리기들을 위해 개발되여야 한다. 그러한 모형은 병렬프로그람의 이식성은 물론 성능, 정확성프로그람화가능성에 영향을 준다.

5. 3. 1. 기억기사건의 순서화

일치성문제는 같은 혹은 각이한 처리기들에 의해 시작된 읽기/쓰기사건들의 각이한 순서화로 인하여 생긴다.

여기서는 기본개념들을 설명하고 실례를 통하여 조종되는 일치성을 얻을 필요를 설

명한다.

공유기억기동작은 프로그람순서와 기억기접근순서에 의해 결정된다.

기본개념 다중처리기에서 병행명령흐름들은(스레드 혹은 프로쎄스들) 각이한 처리기들에서 동시에 실행할수 있다.

한 프로쎄스에 의해 수행된 기억기사건들은 다른 프로쎄스들에 의해 쓰인 자료를 참조할수 있다. 기억기사건들은 공유기억기의 읽기 혹은 쓰기에 대응한다. 기억기일치성모형은 한 프로쎄스에 의해 시작된 기억기사건들이 기계에서 다른 프로쎄스들에 의해 어떻게 관측되여야 하는가를 규정한다.

사건순서화는 몇가지 프로쎄스들이 같은 기억기장소모임접근을 위해 경쟁할 때 어느 기억기접근이 주어 진 시간에 허용되고 어느 기억기가 다음의 접근을 기다려야 한다는것을 선언하는데 쓰일수 있다.

주어 진 일치성모형에 의하여 생성된 기억기순서는 병행프로쎄스들의 정확하고 속도가 빠른 실행을 위하여 다중처리기들의 특정한 순서로 기억기에 접근하도록 지시한다.

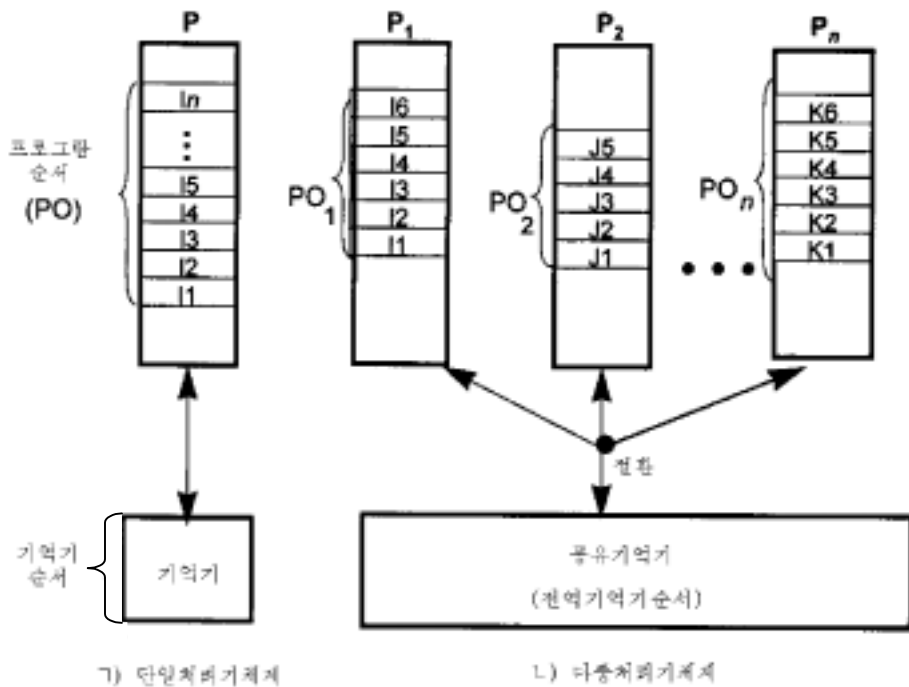



그림 5-8. 단일처리기체계와 다중처리기체계에서 기억기 순서에 대한 프로그람순서

다중처리기에서 기억기사건들 그림 5-8 ㄱ)에서 보여 준바와 같이 단일처리체계는 기억기순서를 결정하는 순차프로그람순서를 따른다. 기억기에 대한 읽기는 언제나 마지막쓰기를 같은 장소에 복귀한다. 그러나 이것은 MIMD다중처리기의 경우는 아니다.

관계되는 병행프로그람들을 실행하는 다중처리체계를 리용하는데서 국부의존성검사는 필요하지만 병행실행의 결과를 유지보존하는데는 충분하지 않을수 있다.

이것을 그림 5-8 ㄴ)에서 보여 준다. 여기서 3개 명령흐름들은(I, J, K) 단일포구절환조종을 통해 공유기억기에 접근하는 읽기/쓰기요청을 발행하기 위해 경쟁한다.

실행결과의 정확성과 예측가능성을 유지하는것은 다음과 같은 리유로 하여 MIMD 다중처리기체계들에서는 매우 복잡하다.

개별적인 명령흐름의 프로그람순서는 그들사이의 호상작용때문에 변경될수 있다. 명령흐름들사이에 동기화가 존재하지 않으면 많은 량의 명령엇끼우기가 가능하다. 최량 전역기억기순서를 찾는것은 어려운 문제이다.

접근들이 캐쉬에 기초한 체계에 존재하는 같은 자료의 다중복사로 많이 일어 나면 각이한 처리기들은 각이한 순서화를 개별적으로 관측할수 있다. 이 경우 다중프로그람의 가능한 동시성의 총체적수는 훨씬 커진다.

실례 5.5. 3-처리기체계에서 기억기사건순서화

공유변수들은 초기에 령값을 가진 모임이다.

매 흐름에서 인쇄지령문이 혼돈을 피하기 위해 같은 주기동안 두가지 변수들을 나누어 지지 않게 읽는다고 가정하자.

전역기억기순서화를 검사하기 위하여 프로그람순서로 P_1, P_2, P_3을 련결시켜 6-조의 2진렬을 형성한다.

그러면 2^6=64개의 가능한 출구결합이 나온다.

모든 처리기들이 자기의 프로그람순서로 명령들을 실행한다면 순서 (a,b,c,d,e,f)는 출구렬 001011을 생성한다. 다른 엇끼우기를 하면 (a,c,e,b,d,f)는 역시 개별적인 프로그람순서를 보존하고 다른 출구렬 111111을 생성한다.

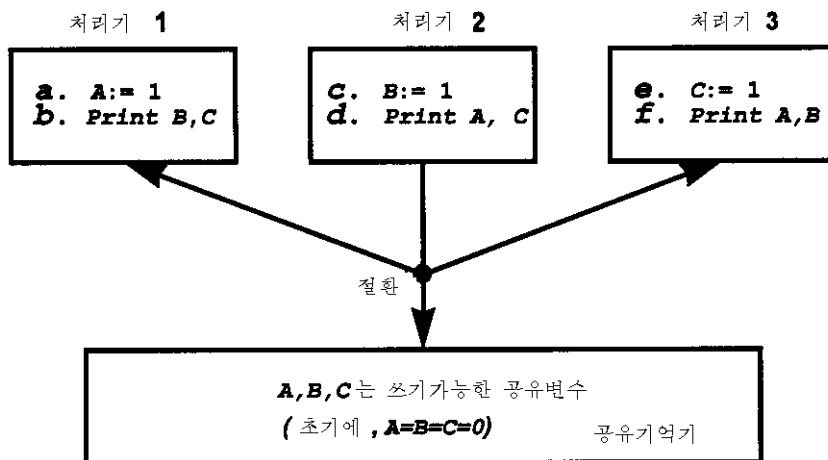



그림 5-9. 3개 처리기들에 의해 비동기적으로 시작된 쓰기 그리고 인쇄동작을 가진 3개 프로그람실행

처리기명령들을 프로그람순서에서 벗어 나서 실행하도록 하면 재순서화된 명령들속에 자료의존성이 존재하지 않는다고 가정할 때 엇끼우기 (b, d, f, e, a, c) 는 출구렬 00000을 생성한 6!=720개의 가능한 순서렬에서 90개만이 개별적인 프로그람순서를 보존한다. 이 90개의 엇끼우기들로부터 모든 6-조결합들이 정확히 실행된다고는 보지 않는다.

실례로 처리기들이 명령들을 프로그람순서대로만 실행한다면 결과 000000는 가능하지 않다.

다른 실례로 결과 011001은 각이한 처리기들이 각이한 순서로 기억기사건들을 관측할수 있으면 허용된다는것을 보여 준다. 이 실례는 기억기일치성을 완화시켜 공유기억기에 접근하는데 거의 제한을 주지 않을수 있다는것을 명백히 보여 준다.

5. 3. 2. 기억기일치성모형

단일처리기 혹은 다중처리기들은 아래에서 특징 짓는바와 같이 순차일치성모형에 의존된다.

약한 일치성, 처리기일치성 그리고 해제(release)일치성공유기억기모형을 얻기 위하여 순차제한을 완화할수 있다.

우의 실례로부터 알수 있는바와 같이 두개의 각이한 처리기의 읽기, 쓰기순서는 큰 차이가 있다. 그러므로 상대적인 사건순서화는 기억기일치성모형을 정의하기 위해 아래와 같이 진행하여야 한다.

순차일치성 이 기억기일치성모형은 모든 처리기들에 의한 읽기, 쓰기, 교체들이 개별적인 처리기들에서 프로그람순서에 따르는 단일전역기억기순서에 따라 직렬로 실행할것을 요구한다. 이것은 명령흐름들이 어떻게 엇끼우기되는가에 관계없이 대역순서는 모든 개별적인 프로그람순서들을 보존하여야 한다는것을 의미한다.

정의 5.1 Lamport[391]은 실행결과가 모든 처리기들의 동작이 어떤 순차순서로 실행되였고 매 개별적인 처리기의 동작은 자기 프로그람에 의해 지정된 순서에 따라 차례로 출현한것과 같다면 다중처리기체계를 순차적일치(SC)체계로 정의하였다.

SC기억기모형은 다중처리기에서 병렬성을 개발할 때 프로그람의 최량화를 요구하지 않는다. 실례로 다음의 프로그람에서 쓰기와 읽기는 두 처리기들에 상주하는 프로그람순서에 제한되지 않는다.

약한 일치성 Dubois, Sxheurich와 Brigg들은[218] 기억기순서를 프로그람의 동기화점과 관련시켜 약한 일치성(WC)기억기모형을 유도하였다. 이 모형은 다음과 같은 3개의 기억기접근조건으로 규정된다.

(1) 이전의 모든 동기화접근은 읽기 혹은 쓰기접근이 어떤 다른 처리기에 관해 허용되기전에 수행되여야 한다.

(2) 이전의 모든 읽기, 쓰기접근은 동기화접근이 어떤 다른 처리기에 대하여 수행되기전에 수행되여야 한다.
(3) 동기화접근은 어떤 다른것에 대하여 순차적으로 일치한다.

이 조건들은 다중처리기에서 기억기접근사건의 약한 순서화를 제공한다. 순차적순서화는 하드웨어-인식동기변수들만으로 제한된다. 두 동기화점들사이의 모든 읽기/쓰기동작은 프로그람순서를 따르지 말아야 한다.

이것은 동기화동작의 부분인 쓰기제외한 재순서화된 쓰기동작의 완충을 위한 기회를 열어 준다.

다중처리기들에서는 기억기접근을 완충시켜 관흐름화된 기억기접근을 가능하게 할수 있다.

처리기일치성 Goodman[273]은 매 개별적인 처리기에 의하여 발행된 쓰기들이 언제나 프로그람순서에 따르는 처리기일치성(PC)모형을 도입하였다.

그러나 각이한 처리기들로부터의 쓰기들은 프로그람순서밖에 있을수 있다. 다시 말하면 쓰기들에서 순차일치성은 매 처리기에서 관측되지만 매 처리기에서의 읽기순서는 다른 처리기들을 포함하지 않는 이상은 제한되지 않는다.

PC모형은 각이한 처리기들로부터 쓰기에 대한 일부 제한을 제거함으로써 SC모형을 완화시켜 얻는다.

이것은 쓰기완충과 관흐름화를 위한 더 좋은 기회를 준다.

다른 처리기들과 관계되는 다음의 두 조건들은 처리기일치성을 담보하기 위해 요구된다.

어떤 다른 처리기에 대하여 읽기가 수행되기전에 이전의 모든 읽기접근들은 수행되여야 한다.

어떤 다른 처리기에 관해 쓰기가 수행되기전에 읽기 혹은 쓰기접근들이 수행되여야 한다.

이 조건들은 읽기가 쓰기를 동반하도록 하여 쓰기를 무시한다.

교착을 피하기 위하여 실행은 프로그람순서로 이전에 출현한 쓰기가 꼭 수행될것이라는것을 담보하여야 한다.

해제일치성 해제일치성(RC)모형은 Gharachorloo 등에 의해 도입되였다[264]. 해제일치성은 프로그람에서 동기화접근들이 획득(lock, 차단) 혹은 해제(unlock, 비차단)로 식별될것을 요구한다.

획득(acquire)은 읽기동작(읽기-변경-읽기의 부분으로 될수 있다.)인데 자료모임에 접근하는 허가를 준다.

해제는 그러한 허가를 포기하는 쓰기동작이다.

이 정보는 동기화점들사이의 기억기접근을 완충화하고 관흐름화하는 유연성을 보장하는데 쓰인다.

완화형기억기모형의 기본우점은 가능한 많은 쓰기지연시간을 은페함으로써 더 좋은

기억기성능을 얻는것이다. 기본결합은 보통 사용자들에게 하드웨어복잡성과 프로그람작성모형의 복잡성을 증대시킨다는것이다.

다음의 세가지 조건들은 RC공유기억기모형을 담보한다.

- 보통의 읽기 혹은 쓰기접근이 어떤 다른 처리기에 대하여 허용되기전에 이전의 모든 획득접근들을 수행하여야 한다.
- 해제접근이 어떤 다른 처리기에 대하여 수행하는것이 허용되기전에 이전의 모든 보통읽기와 기억접근들을 수행하여야 한다.
- 전용접근은 다른것과 처리기에서 일치한다. 약한 일치성에 의한 순서화제한들은 해제일치성에서는 존재하지 않는다. 대신 RC모형은 처리기일치성을 요구하지만 순차일치성은 요구하지 않는다.

Nltzberg와 Lo[468]은 우의 4개 기억기일치성모형들사이의 관계를 그림 5-10에서와 같이 특징 지었다.

RC모형은 그림 5-10에서 보여 준바와 같이 PC와 WC모형의 우점들을 결합하였다. RC기억접근은 그것이 완료될 때까지 처리기가 획득접근하는것을 막는 (1)에 의해 규정될수 있으며 이전의 모든 접근들이 완료될 때까지 해제접근완료를 지연시키는 (2)에 의해 만족될수 있다. SC에 비한 RC모형실행비용은 차단해제를 제공하고 미해결된 다중요청들의 추적을 보존하는데 필요되는 외부하드웨어/쏘프트웨어로부터 생긴다. 이 비용은 비록 무시할수 없지만 같은 특징들이 미리꺼내기와 다중문맥을 지원하는데 역시 요구된다.

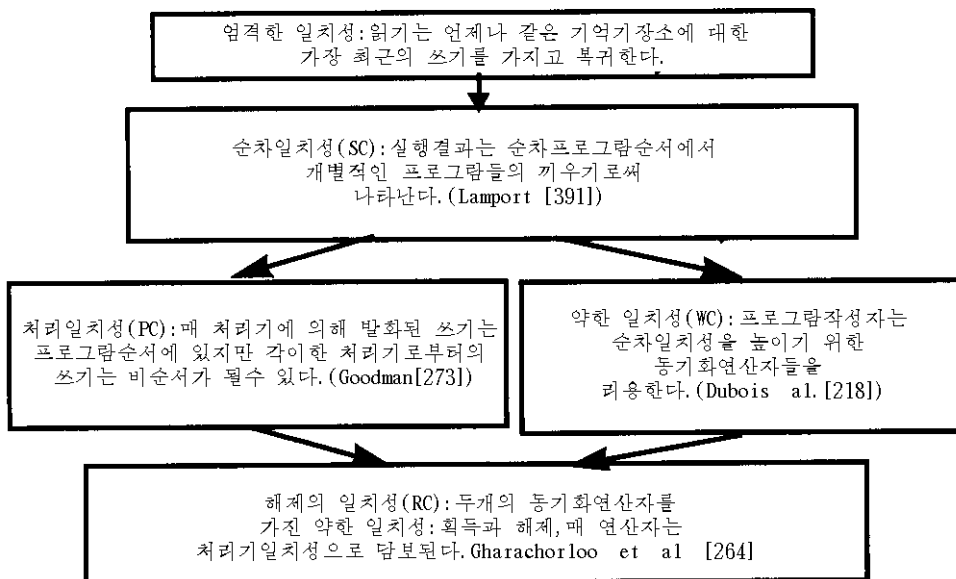



그림 5-10. 4개의 기억기일치성모형들의 직관적정의들

5. 3. 3. 완화형기억기모형

완화형기억모형의 기본목표는 캐쉬일관성조종의 부가처리를 완화시키는것이다. 일관성부가처리에서 감소는 프로그람실행의 정확성을 유지해야 한다. 완화형기억모형은 다음과 같은 두가지 특성을 가지고 있다([8]).

- 기억모형은 두개의 쓰기사이에 쓰기로부터 쓰기에 뒤따르는 읽기로 그리고 읽기로부터 읽기로 혹은 읽기에 뒤따르는 쓰기로의 순서를 어떻게 완화시키는가 하는데 기초하여 차이가 생긴다. 이 완화는 각이한 주소를 가진 동작쌍에만 적용하며 비캐쉬구성방식의 순차일관성에 대한 최량화와 류사하다.
- 어떤 기억모형들은 쓰기가 다른 모든 처리기들에 명백하게 진행되기전에 읽기가 다른 하나의 처리기의 쓰기값을 복귀하게 한다.

이 완화는 캐쉬에 기초한 체계에만 적용한다.

상품화된 기억모형들 완화형기억기일치성모형들은 각이한 장소에 대한 모든 동작들사이의 프로그람순서를 완화시켜 읽기 혹은 쓰기가 뒤따르는 읽기 혹은 쓰기에 대하여 재순서화하도록 한다. 따라서 그것들은 순차일치성을 위반할수 있다.

읽기동작다음의 기억기동작들은 읽기에 대하여 중복되거나 재순서화될수 있다. 이 우연성은 하드웨어가 읽기지연시간을 정적(순서) 혹은 동적(비순서)으로 일정짜기된 처리기로 은페시키게 한다.

6개의 완화형기억기모형들을 표 5-4에 제시한다.

그 6개모형을 보면 약한 순서화(WO)모형, 두개의 해제기억모형(RC_{sc}와 RC_{pc}) 그리고 Digital Alpha, Sparc RMO, IBM Power PC에서 리용한 3개의 모형들이다. Alpha를 제외한 이 모형들은 역시 같은 장소에 대한 두개의 읽기를 재순서화하여야 한다.

표에서 X는 주어 진 기억기모형의 실행에서 완화가 허용된다는것을 가리킨다. 또한 완화가 프로그람결과에 트랜잭션함으로써 프로그람작성자에 의해 검측될수 있다는것을 가리킨다.

그러나 읽기-own-쓰기-early완화는 SC, WO, Alpha 혹은 Power PC모형들로는 검측되지 않는다.

읽기-others-쓰기-early완화는 가능하며 RC_{sc}모형의 복합실행으로 검측될수 있다. 공백칸은 접근제한들이 주어 진 기억기모형에 적용되지 않을 경우들에 대응한다.

완화형기억일치성모형은 순차일치성모형보다 더 좋은 성능을 얻기 위해 설계되였다. [8]에 따르는 기억기와 통신속도에 관계되는 처리기속도에서의 증가는 이 모형들로부터 가능한 리득을 얻는다. 하드웨어성능에서 리득을 보는것외에 완화기억기일치성모형들은 콤파일러최량화를 가능하게 하는데서 중요한 역할을 한다. 이러한 리유들로부터 Digital Alpha, Sun Sparc 그리고 IBM Power PC 등과 같은 많은 상업적구성방식들은 표 5-4에서와 같이 완화형일치성을 지원한다.

매개의 자세한 해석은 표 오른쪽 마지막란에 있다.

표 5-4 여러가지 완화형기억일치성모형들

해제 기억기	W_ _>R Order	W_ _>W Order	R_ _>RW Order	읽기-Others-쓰기-Early	읽기-Own-쓰기-Early	보안 Nets Ref.
SC	X				X	No[391]
IBM370	X					[341]
TSO	X				X	[539]
PC	X			X	X	[264]
PSO	X	X			X	[579]
WO	X	X	X		X	[218]
RCsc	X	X	X		X	Nsync, [264]
RCpc	X	X	X	X	X	
Alpha	X	X	X		X	MB,WMB,[563]
RMO	X	X	X		X	Variou MEMBARs ,[579]
PowerPC	X	X	X	X	X	Sync[434]
대표적인 콤퓨터 체계	AlphaServer 8400, Cray T3E, NUMA-Q, Sparc-Center 2000, Convex SPP, Ultra Servers	AlphaServer 8400, Cray T3D/T3E, Convex SPP in WO mode		Cray T3D	AlphaServer 8400, Cray T3D/T3E, SparcCenter 200, Ultra Ent, Servers	

보안망들은 암묵적완화들을 무효로 하는 기구를 가리킨다.

실례로 read-변경-write화, 여러가지 동기화동작들과 같은 명백한 방어명령들은 프로그람순서완화를 무시하는데 쓰일수 있다.

캐쉬일관성규약과의 관계 캐쉬일관성규약은 주어 진 기억기일치성모형에 의한 기억기접근제한을 관측하여야 한다.

실례로 처리기캐쉬에서 읽기적중을 고찰하자.

이전의 쓰기동작완료를 기다림이 없이 캐쉬된 값을 읽는것은 순차일치성을 위반할수

있다. 캐쉬일관성규약은 새롭게 씌여 진 값을 방송하는 기구를 제공한다.

기억기일치성모형은 값이 주어 진 처리기로 방송될 때 보충적인 제한을 준다.

쓰기완료의 검측은 결합된 캐쉬규약/기억기일치성모형의 성공적인 실행에서 결정적인 역할을 한다.

쓰기는 결국 모든 처리기들에서 명백하여야 한다. 같은 장소에 대한 모든 쓰기는 순차일치성을 높이기 위하여 모든 처리기들에서 같은 순서로 되여야 한다. 프로쎄스를 갱신하는 다중캐쉬는 고유하게 작지 않은 기억기들이다.

쓰기를 매우 작게 유지하는것은 또 하나의 중요한 요구이다.

이러한 론점들은 실지 체계설계들에서 완성되고 있다.

콤파일러지원 이것은 최량화의 또 다른 준위를 제공한다. 기본방법은 주어 진 캐쉬규약/기억기일치성도식에 의해 주어 진 접근제한을 만족시키기 위하여 기억기동작들을 재순서화하는것이다.

- 첫째로, 콤파일러는 공유기억기동작들가운데서 프로그람순서를 보존하여야 한다.
- 둘째로, 콤파일러에 의한 재순서화는 일치성모형에 의해 주어 지는 접근제한을 위반할수 없다. 사실상 이 요구들은 하드웨어에 의한 재순서화에 적용한다.

주어 진 기억기일치성모형을 지원하기 위한 명시적병렬성콤파일러는 암시적병렬성콤파일러작성보다 훨씬 쉽다.

현재 암시적인 병렬성을 가진 완화형기억기일치성을 지원할수 있는 상업화된 콤파일러는 없다.

5. 4. 분산캐쉬/기억기구성방식

물리적으로 분산된 기억기들은 론리적으로 공유되거나 공유되지 않을수 있다. 과학과 공업에서 쓰이는 공유기억기라는 용어는 서로 다른 의미를 가질수 있다. 혼돈을 피하기 위하여 공유기억기개념을 명백히 해야 한다.

먼저 공유기억기구성방식을 공유기억기프로그람작성환경과 구별해야 한다.

공유기억기구성방식(다중처리기)은 공유기억기와 통보문넘기기프로그람작성모형을 지원할수 있다. 공유기억기프로그람작성구성모형은 공유기억기 혹은 비공유기억기구성방식(실례로 다중콤퓨터)에서 실행될수 있다. 그것은 일반적으로 SMPs는 공유기억기구성방식을 가지고 일반적인 MPPs는 가지지 않는것과 일치한다. 바로 그사이에 많은 분산기억기구성방식이 있다. 이 분산공유기억기체계들은 완전히 다른 구성방식과 프로그람적인 특징들을 가지고 있다.

5. 4. 1. NORMA, NUMA, COMA, DSM 모형

여러가지 다중처리기들과 다중콤퓨터들을 그림 5-11에 분류하였다. 이 나무분류는 모든 기억기모형들에 기초하고 있다. 먼저 사용자의 견지에서 몇가지 일반적인 경우들을 관찰하여 공유기억기구성방식술어를 명백히 하자.

어떤 처리기에서 동작하는 프로쎄스가 전체 체계에서 국부 혹은 원격기억기에 직접 접근할수 있으면 체계는 공유기억기구성방식을 가진다.

반대인 경우 비공유기억기구성방식(다중콤퓨터)을 가진다. 여기서 직접이라는것은 적재 혹은 기억과 같은 명령이 전체 다중처리기체계에서 어떤 기억기장소에 접근할수 있다는것을 의미한다.

기억기공유는 비공유기억기구성방식에서 가능하다. 그러나 원격기억기접근은 직접이 아니지만 실행시간 혹은 사용자호출가능한 서고와 같은 쏘프트웨어층을 통하여 직접이 가능하다.

이것은 접근지연시간에서의 차이로 넘어 간다.

현재 다중처리기들은 수백ns의 원격접근지연시간을 가지며 다중콤퓨터들은 수십㎲의 원격접근지연시간을 가진다.

중심기억기 대 분산기억기 병렬콤퓨터는 중심기억기나 혹은 분산기억기구성방식을 가진다. 분산기억기체계들은 비동일기억기접근(non-uniform memory access, NUMA)과 비원격기억기접근(non-remote memory access, NORMA)의 구성방식을 포함한다.

일부 사람들은 이 체계들에서 기억기접근들이 역시 비동일하게 진행되므로 NORMA를 포함하는 보다 넓은 정의인 NUMA를 리용한다. 중심기억기체계들은 역시 동일기억기

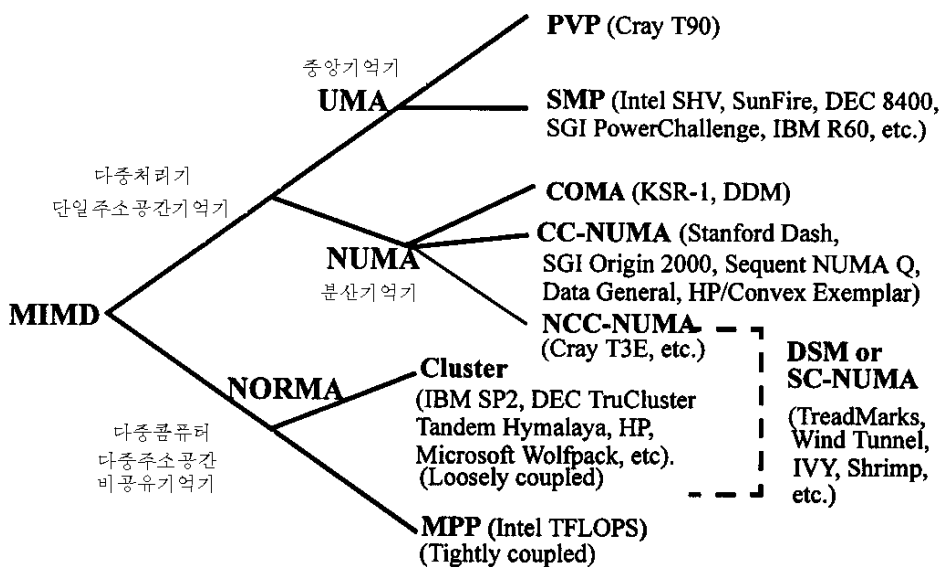



그림 5-11. 병렬, 분산 그리고 클라스터콤퓨터제작을 위한 여러가지 형태의 기억기구성방식

접근(uniform memory access,UMA)체계로 알려 져 있다.

UMA구성방식에서 모든 기억기장소들은 처리기로부터 같은 거리에 있으며 모든 기억기접근들은 대체로 같은 시간이 걸린다. UMA체계는 두가지 형태 즉 병렬벡토르처리기(PUP)와 대칭다중처리기(SMP)가 있다.

PVP를 벡토르초고속콤퓨터라고도 부른다.

분산기억기구성방식 분산기억기콤퓨터는 다중마디를 포함하는데 매개는 하나 또는 그이상의 처리기들과 국부기억기를 가진다. 다른 마디들에서는 기억기들을 **원격기억기들**이라고 부른다.

몇가지 형태의 분산기억기구성방식들은 현재 상업용 그리고 연구용병렬체계들에서 쓰인다. 그것들을 보면 NORMA, 비캐쉬일관성 NUMA(NCC-NUMA), 캐쉬일관성 NUMA(CC-NUMA) 그리고 캐쉬기억기구성방식(COMA) 등이다.

이것들의 기본차이를 그림 5-12에 제시한다.

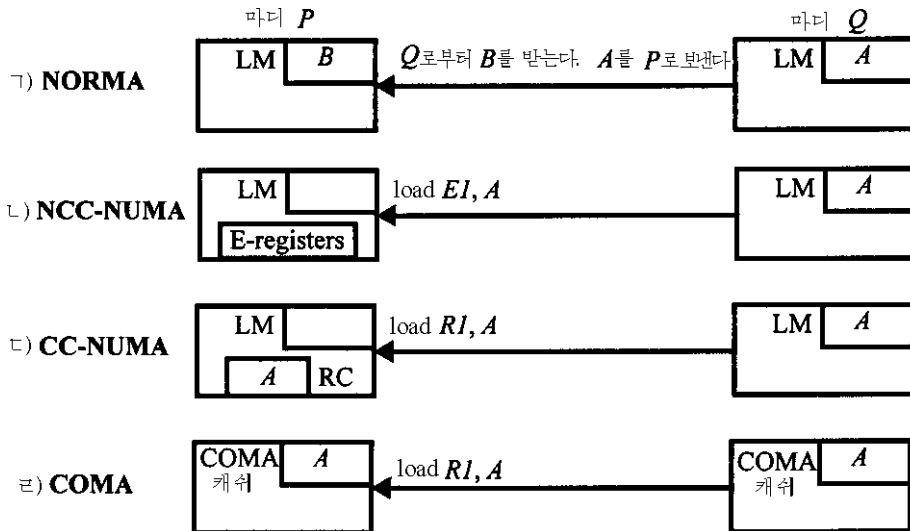



그림 5-12. 4개 분산기억기구성방식의 비교
(LM: 국부기억기, A, B: 기억기주소, RC: 원격캐쉬,
E1: E등록기, R1: 일반목적등록기)

NORMA모형은 이미 1.3.4절에서 고찰한바와 같이 성긴결합형(loosely-coupled)클라스터와 밀집결합형(tightly-coupled) MPPs로 더 나눈다.

클라스터들과 MPPs는 10장과 11장에서 각각 자세히 고찰된다. NORMA기계에서 마디기억기들은 개별적인 주소공간을 가진다.

마디는 원격기억기에 직접 접근할수 없다. 원격자료에 접근하는 방식은 통보문을 보내는것뿐이다. 그림 5-12 ㄱ)에서 마디 P는 마디 Q의 자료 A를 가지기를 원한다. 이것은 마디 Q가 보내기 routine을 실행하고 마디 P는 대응하는 받기 routine을 실행하는것으로 완성된다.

마지막으로 A의 값은 마디 P의 국부기억기에 있는 변수 B에 복사된다. 다른 3가지 구성방식들은 모두 단일주소공간에 모든 국부기억기들을 붙이는 전용하드웨어를 가지고 있어 처리기가 기억기장소에 대한 접근을 가능하게 한다.

실례로 그림 5-12에서 보여 주는바와 같이 원격자료 A를 얻기 위하여 적재명령을 리용할수 있다. 그러나 그것들의 접수기구는 차이난다.

전형적인 NCC-NUMA기계는 Cray T3E이다. 국부기억기외에 매 마디는 E등록기라고 부르는 마디준위등록기모임을 가진다(그림 5-12 ㄴ). 명령은 값 A를 E등록기 E_1에 적재한다. 그다음 값은 처리기등록기 혹은 국부기억기에 전송될수 있다.

다른 NCC-NUMA체계들은 원격값을 직접 처리기등록기로 적재하는것을 허용할수 있다(즉 명령 "load R1, A"에 의해). 캐쉬블로크 A는 국부처리기캐쉬 혹은 국부기억기로 자동적으로 복사되지 않는다.

CC-NUMA체계에서 명령은 값 A를 국부처리기등록기 R1(그림 5-12 ㄷ)에 적재한다. 동시에 캐쉬블로크 A는 자동적으로 원격캐쉬(RC)라고 하는 마디준위캐쉬에 복사된다. 그러나 이 캐쉬블로크는 국부기억기로 복사되지 않는다.

어떤 CC-NUMA체계들은 원격캐쉬들을 가지지 않는다(실례로 SGI Origin2000). 그다음 캐쉬블로크 A는 마디의 2차캐쉬로 복사된다.

COMA기계에서 모든 국부기억들은 캐쉬들로 구조화된다(COMA캐쉬라고 부른다.). 그러한 캐쉬는 2차캐쉬나 마디의 원격캐쉬보다 훨씬 큰 용량을 가진다. 명령은 값 A를 국부처리기등록기 R1로 적재한다(그림 5-12). 동시에 캐쉬블로크 A는 국부기억기(현재 COMA캐쉬)에 자동적으로 복사된다. COMA는 단지 다중국부기억기들에 같은 캐쉬블로크를 복제하기 위한 하드웨어지원을 제공하는 구성방식이다.

NCC-NUMA와 CC-NUMA/COMA NCC-NUMA체계는 캐쉬일관성을 위한 하드웨어 지원을 가지지 않는다. CC-NUMA와 COMA구성방식은 하드웨어에 의한 캐쉬일관성지원을 제공한다. 이러한 리유로 CC-NUMA나 COMA체계보다 확대가능한 NCC-NUMA체계를 제작하는것이 더 쉽다.

잘 알려 진 실례로는 Cray T3D/T3E체계를 들수 있는데 전체적으로 주소공간일관성을 유지하는 쏘프트웨어에 의거한다. 이것은 다음과 같은 문제를 야기시킨다.

구성방식을 자세히 고찰하면 체계쏘프트웨어와 그것의 리용은 주소공간을 관리하는 유연성을 가지고 성능을 최대화한다.

그러나 대부분의 사용자들은 그러한 자세한것을 보려고 하지 않는다. 체계쏘프트웨어가 일관성주소공간을 유효하게 관리하도록 하는것은 쉽지 않다. Cray T3E는 절충안을 선택한다. 성능지향의 사용자들을 위해서 T3E는 사용자가 자료공유를 최량으로 관리하는 낮은 준위공유기억기서고를 제공한다. 사용자들의 편리성을 위하여 T3E는 높은 준위 프로그람작성도구를 제공한다. 그러면 콤파일러/실행시간체계는 기억기일관성을 유지한다. 공업적체계들은 SGI Origin 2000, HP/Convex Examplar X-Class, Sequent NUMA-Q2000 그리고 자료일반다중봉사기 NUM체계들을 포함한다.

일부 상품화된 CC-NUMA체계들을 8장에서 고찰한다.

COMA모형은 Kendell Square KSR-1/2체계들[116]과 콤퓨터과학 Swedish연구소에서

개발된 자료발산기계(DDM)[294]에서 실행되였다.

KSR는 1에서 3준위의 계층으로 제작된 다중처리기이다. KSR기계에서 기억기구성방식을 ALLCAHE라고 부르는데 모든 국부기억기들이 캐쉬로 구성된다.

CC−NUMA와 COMA CC−NUMA다중처리기에서 주기억기는 모든 국부기억기들로 이루어 진다. COMA다중처리기에서 주기억기는 모든 COMA캐쉬들로 이루어 진다. COMA구성방식은 COMA캐쉬들을 위한 태그들과 상태정보를 유지하는 하드웨어지원을 요구한다. 같은 하드웨어는 COMA캐쉬관리를 가상기억기관리와 통합하여야 한다(즉 페지를 디스크로부터 COMA캐쉬들로 어떻게 가져 오는가?). 이 모든 복잡성은 COMA체계가 UNMA기계보다 실행되는데 더 많은 비용을 들게 한다.

프로그람이 CC−NUMA에 적재되여 실행할 때 조작체계는 매 캐쉬행을 홈마디에 배정하는데 이것은 국부기억기와 캐쉬행이 배당되고 영원히 상주하는 곳이라는것을 의미한다.

마디 P가 캐쉬행에 접근할 때 P는 일시적으로 캐쉬행을 P의 국부캐쉬에 기억한다[그림 5−12 ㄷ)].

WB시간에서 캐쉬행은 홈마디의 국부기억기에 뒤로 기억된다. 그러한 후쓰기통행을 완화하기 위하여 CC−NUMA조작체계는 복제하거나 페지가 가장 많이 참조되는 마디의 국부기억기로 페지를 이동할수 있다.

실례 8.2를 통해서 자세히 보기로 하자.

COMA와 CC−NUMA의 기본차이는 COMA가 복제와 이동을 더 효과적으로 조종한다는것이다.

첫째로, COMA는 복제와 이동을 실현하기 위해 하드웨어를 리용한다. COMA기계를 리용하기 위하여 자료는 아무곳에나 배당될수 있다. 실행시간에 캐쉬행은 필요되는 곳에로 이동할수 있다.

둘째로, 복제나 이동의 립도는 COMA에서 캐쉬행이며 CC−NUMA에서의 페지립도와 비교된다.

이것은 거짓공유(false sharing)가 CC−NUMA에서보다 COMA에서 그다지 엄하지 않다는것을 의미한다.

거짓공유는 두개 마디가 같은 캐쉬행 혹은 같은 페지에 상주하는 두개의 서로 다른 자료변수들에 접근하는 상황을 가리킨다.

총체적으로 CC−NUMA와 COMA구성방식의 성능은 두개의 기본인자들에 달려 있다. 즉 작업모임의 크기와 통신 대 계산률(communication−to−computation, CCR)이다.

낮은 CCR와 작은 작업모임에 대하여 두 구성방식들은 성능을 잘 수행한다. 왜냐하면 대부분의 기억기참조들은 국부처리기캐쉬들에 의해 만족되기때문이다.

큰 작업모임과 높은 CCR에 대하여 두 구성방식들은 자기의 성능을 불충분하게 수행한다. 이것은 큰 작업모임이 두 구성방식에서 더 많은 비적중을 일으키고 높은 CCR는 주어 진 응용프로그람에서 더 큰 통신요구를 의미하기때문이다.

높은 CCR와 작은 작업모임에 대하여 CC−NUMA기계는 COMA가 더 높은 비적중벌칙을 가지기때문에 COMA기계보다 더 잘 수행한다. 낮은 CCR와 높은 작업모임에 대하

여 COMA기계는 CC−NUMA기계보다 더 잘 수행된다. 왜냐하면 COMA캐쉬들이 CC−NUMA에서의 원격캐쉬보다 훨씬 큰 용량을 가지기때문이다. 이 결과들은 Stenstrom 등의 해석에 기초한다[585].

쏘프트웨어실행형DSM NORMA와 NCC−NUMA에서 공유기억기계산을 가능하게 하기 위하여 연구자들은 쏘프트웨어−일관성 NUMA(SC−NUMA)기억기모형(분산공유기억기(DSM)모형으로도 알려 져 있다.)을 제기하였다. 일부 DSM기계들은 10장에서 취급한다.

쏘프트웨어실행형DSM은 단일주소공간, 자료공유 그리고 일관성규약을 주는 쏘프트웨어확장에 주로 의거한다.

공유가상기억기(SVM)방법이 있는데 전통적인 마디조작체계에서 가상기억기관리기구는 페지준위에서 자료공유를 제공하도록 변경된다.

조작체계를 변경시키지 않는 방법이 있다. 그것은 단일주소공간코드를 다중주소공간에서 수행하도록 변환하는 콤파일러와 서고함수들을 리용한다. 응용코드들은 자료공유, 동기화 그리고 원시일관성을 포함하도록 변경되여야 한다.

실례 5.6. 공유기억기계획작성에서 절충

공유기억기를 언급할 때 콤퓨터판매자들과 사용자들은 흔히 여러가지 일들을 생각한다. 회사가 4개 마디와 4GB의 기억기를 가진 기계를 팔고 있다고 가정하자. 판매표본은 체계가 연산체계와 응용에 의해 모든 기억기들이 리용될수 있는 공유기억기기구를 제공하는 특징을 가지고 있다.

사용자는 아주 많은 량의 수값계산을 위한 수학적쏘프트웨어를 동작시킬수 있는 3GB 기억기를 요구하기때문에 현지에서 기계를 살수 있다. 그러나 그는 공유기억기기구가 자기에게 싫든 좋든 다음의 어느 한 문제를 실행하라고 하는것을 보고 곧 실망할수 있다.

(1) 그 응용으로 판매자가 제공하는 실행시간서고를 련결시킨다.
(2) 판매자가 공급하는 콤파일러를 리용한 그것의 응용을 재콤파일러한다.
(3) 그것의 응용에 서고함수호출을 삽입하고 재콤파일러와 재련결편집을 한다.
(4) 그 응용재쓰기하는 새로운 프로그람작성언어를 리용한다.

사용자는 수학의 대상코드를 가지고 있지 않기때문에 과제 (1)를 실행할수 없다. 그는 수학원천코드를 가지고 있지 않기때문에 과제 (2)~(4)를 실행할수 없다. 원천을 가지고 있다고 해도 너무 시간을 소비하므로 과제 (3) 혹은 (4)를 실행하기를 원하지 않을수 있다.

그는 바로 생소한 언어구조나 서고함수를 리용하는 익숙치 못한 기계를 계획하는것이 아니라 자기 일을 원만히 진척시켜 나가기를 원한다.

수학을 쏘프트웨어판매자에 의해 기계에 내장하면 우의 문제들은 나타나지 않을것이다.

이 실례가 보다 넓은 사용자의 찬성을 얻기 위해서는 체계판매자는 제3자의 응용을 포함한 응용을 허용해야 한다. 체계가 진짜 공유기억기라면 실천적인 검사기준은 순차2

진코드가 모든 기억기를 리용하게끔 할 때 눈에 보인다. 오늘날의 상업체계에서 UMA와 CC−NUMA체계만이 이 공유수준에 도달할수 있다. 이 기준에 맞는 효과적인 SC−NUMA를 어떻게 설계하는가는 아직 연구중에 있다.

분산기억기개괄 표 5−5에서 5개 분산기억기구성방식을 종합고찰한다. 일관성은

표 5-5 5개 분산기억기구성방식의 특징

속성	NORMA	NCC-NUMA	SC-NUMA	CC-NUMA	COMA
립 도	대상	대상	페지	캐쉬-기준	캐쉬-기준
거짓공유	아니	아니	예	예	예
재현과 이동	대상준위에서 사용자에 의해 수행	대상준위에서 사용자에 의해 수행	OS 혹은 페지준위에서 실행시간에 의해 수행	페지준위에서 OS에 의해 수행	캐쉬기준준위에서 하드웨어에 의해 수행
일관성	사용자응용에 의해 수행	사용자준위 쏘프트웨어에 의해 수행	OS 혹은 실행시간에 의해 수행	캐쉬기준준위에서 하드웨어에 의해 수행	캐쉬기준준위에서 하드웨어에 의해 수행

CC−NUMA와 COMA에서 하드웨어, SC−NUMA에서 OS나 실행시간서고 그리고 NORMA 혹은 NCC−NUMA에서 사용자준위쏘프트웨어에 의해 시행된다.

일반적으로 NORMA와 NCC−NUMA체계들은 대상(변수)준위에서 자료를 공유한다. 마디는 어떤 변수이든지 필요하면 가진다.

거짓공유는 없으며 공간국부성은 리용되지 않는다.

COMA와 CC−NUMA체계들은 보통 페지준위에서 자료를 공유한다. SC−NUMA체계들은 보통 페지준위에서 자료를 공유한다. 마지막 3개 형태의 체계들은 어느 정도의 거짓공유를 가지는 대신에 공간국부성을 리용할수 있다.

자료배당, 복제 그리고 이동은 자료대상립도에서 NUMA와 NCC−NUMA의 사용자쏘프트웨어에 의해 수행된다. 복제와 이동은 페지준위에서는 CC−NUMA의 조작체계에 의해서, 캐쉬행준위에서는 COM의 하드웨어에 의해서 그리고 대상 혹은 페지준위에서는 SC−NUMA의 조작체계 혹은 실행시간서고에 의해서 수행된다.

5. 4. 2. 등록부에 기초한 일관성규약

Snoopy규약은 기억기모선에서 방송능력에 기초한다. 다른 캐쉬일관성규약들은 방송을 리용하지 않을수 있다.

등록부에 기초한 규약들은 Snoopy모선에서 방송을 리용하기 위해 설계되지 않았다.

이 절에서는 등록부에 기초한 규약의 기본개념들을 소개한다. 등록부에 기초한 실례를 이 절에서 보기로 한다.

캐쉬등록부 등록부방법은 공유자료의 모든 캐쉬행들의 장소와 상태들을 기록하는 등록부를 리용하는것이다.

이 캐쉬장소들의 목록을 **캐쉬등록부**라고 부른다.

자료의 매 캐쉬행에 대한 등록부입력자료는 같은 기준에서 모든 원격복사들의 장소들을 지정하는 포인터들을 포함한다. 매 등록부입력자료는 유일한 캐쉬가 자료의 련관기준에 쓰기를 허용하겠는가를 지정하는 오물비트를 포함한다.

Tang[606]은 처음으로 캐쉬일관성조종을 위한 등록부도식을 제기하였다. 기본방법은 모든 캐쉬조건들을 기록하는 중심등록부를 리용하는것인데 종래정보와 모든 캐쉬행상태들을 포함한다.

이 중심등록부는 중심화된 공유기억기를 가진 소규모 SMP에서 캐쉬일관성조종에만 적합하다.

기본난점은 중심등록부 그자체를 실행하는데 많은 기억기공간이 요구되고 갱신시간을 줄이기 위해 련관적으로 탐색되여야 한다는것이다.

이 도식은 큰-캐쉬 SMP, NUMA 혹은 분산기억기가동환경에서의 리용에는 적합치 않다.

분산등록부들 분산등록부도식은 Censier와 Feautrier[135]에 의해 제기되였다.

매 기억기모쥴은 상태와 모든 캐쉬행들의 존재정보를 기록하는 개별등록부를 보존한다. 등록부입력자료들은 같은 캐쉬행복사를 가진 원격캐쉬장소를 가리키는 존재정보와 캐쉬행의 상태들을 포함한다. 다음실레는 분산캐쉬등록부들의 기본사상을 명백히 해준다.

실례 5.7. 캐쉬일관성조종을 위한 분산등록부들

그림 5-13에서 공유기억기는 다중기억기모듈로 이루어 진다.

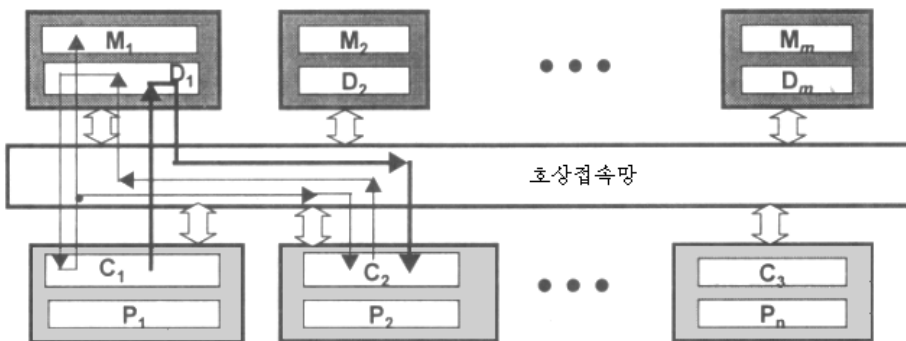



그림 5-13. 등록부에 기초한 캐쉬일관성도식

매 기억기모듈 M_i, $i=1, 2, \cdots, m$은 캐쉬등록부 D_i를 보존한다. 캐쉬들은 처리기 P_i에 대하여 C_i로 표시한다. 실례는 캐쉬 C_1과 C_2사이의 같은 캐쉬행의 공유를 보여 준다. 호상접속망은 모선에 조금도 제한되지 않는다. 여러 단계 혹은 크로스바절환기 혹은 접대접체계령역망(SAN) 혹은 국부령역망(LAN)이 리용될것이다. 이 절환들과

망들은 6장에서 취급된다. 실선(화살선)들은 캐쉬일관성조종에 대한 신호경로를 가리키고 굵은 화살선(속이 비여 있는)은 자료경로를 가리킨다.

기억기/캐쉬갱신은 다음과 같이 동작한다. 캐쉬 C_2에서 읽기비적중(그림 5-13에서 얇은 선)은 C_1에서 clean봉사의 존재를 가리키는 등록부 D_1로 보낸 응답에 귀착된다.

기억기조종기는 C_1로 응답을 다시 넘겨 준다. 이 캐쉬는 그 clean복사를 M_1과 C_2로 되돌려 보낸다. C_1에서 쓰기-적중 (굵은 선)인 경우 지령을 기억기조종기에 보내며 등록부 D_1에서 존재비트로 표시된 모든 캐쉬들에 무효를 보낸다.

캐쉬등록부구조. 각이한 형태의 등록부규약들은 3가지로 분류할수 있다.
즉 full-map등록부, limited등록부, chained등록부

- full-map캐쉬등록부는 대역기지에서 공유된 모든 캐쉬행들의 정보를 포함한다. 매 등록부입력자료는 N개의 포인터를 포함하는데 N은 처리기수이다. 포인터들은 비트벡토르에 의해 식별된다. Full등록부는 매 마디에서 반복하면서 많은 기억기공간을 차지한다. 소규모다중처리기 혹은 다중콤퓨터는 full-map등록부방법을 리용할수 있다.
- limited캐쉬등록부는 체계크기에 관계없이 등록부입구단마다 훨씬 감소되고 고정된 수의 포인터들을 리용한다. 이것은 캐쉬등록부들에서의 기억기요구를 감소할것이며 따라서 기억기공유가 낮지만 실행하는데서 더 경제적이다. 그러나 이것은 기억기공유화정도가 지나치면 캐쉬/기억기갱신처리를 느리게 할수 있다.
- Chained캐쉬등록부는 등록부정보를 작은 국부등록부들에 분배함으로써 full-map 도식과 경쟁한다. 기억기공유의 대역적표상을 얻기 위하여서는 련결된 캐쉬등록부목록을 통하여 탐색하여야 한다. IEEE SCI표준은 chained등록부들의 구조를 규정하였다.

5. 4. 3. Stanford Dash 다중처리기

이것은 Stanford대학에서 제작한 실험용CC-NUMA다중처리기체계이다. 이름 Dash는 공유기억기를 위한 등록부구성방식(directory architecture for shared memory)의 략자이다.

그 의의는 분산일관성캐쉬와 분산계층기억기를 리용하여 단일주소공간을 가진 확대가능한 병렬콤퓨터를 제작하는것이 가능하다는것이다.

Dash개발자들은 이 방법을 CC-NUMA구성방식제작에 도입하여 통보문넘기기다중콤퓨터의 확대가능성을 보존하게 하였다.

Dash Dash구성방식의 하드웨어조직을 그림 5-14 ㄱ)에 제시한다. 그것은 16개의 SGI SMP마디들에 64개까지의 극소형처리기들을 통합한것이며 매개는 4MIPS

R3000/R3010처리기이다. 마디구성방식은 Silicon Graphics, 4D/340 Power Station과 조금 차이난다.

두개 전용보조판(daughter board)이 매 Power Station마디에 꽂기 위해 제작되였는데 이 판들은 망대면회로와 같은 Snoopy모선에 붙은 4개의 모든 처리기들에 의해 공유된 캐쉬등록부를 가지고 있다. 16개 SGI마디들가운데서 호상접속망은 wormhole-routed이고 2차원 4×4그물(mesh)망의 쌍이다. Mesh는 flat wire strip와 wormhole router로 제작되였다.

그물망에서 통로폭은 50ns의 실패에 의한 되돌이(fall-through)시간과 35ns의 주기시간을 가진 16b이다.

하나의 요청그물망은 원격기억기를 요청하는데 쓰고 다른것은 응답그물망이다. 그물망교차에서 작은 4각형들은 Charler Seitz에 의해 개발된 wormhole router들이다.

전용하드웨어 비록 원형이 매 4D/340마디의 기억기주소공간에 대한 제한으로 16마디클라스터(4×4 mesh)로 제한되지만 대수설계자들은 대수방법을 위한 확대가능성을 주장하였다. 리론적으로 체계는 수백개의 처리기들을 지원하기 위해 확대가능하여야 한다. Dash에서 4D/340을 리용하기 위해 현재체계판을 좀 변경시켜 등록부기억기와 내부마디통보문넘기기를 지원하는 새로운 판을 설계하였다.

현재 CPU판에 대한 기본변경은 요청이 원격마디에서의 봉사를 요구할 때 쓰이는 모선재검사신호를 부가한것이였다.

중심모선조정기는 등록부에서 마스크를 접수하도록 변경되였다. 마스크는 원격요청이 봉사될 때가지 처리기의 재검사를 지체시킨다. 이것은 원격봉사의 요청을 위한 분할처리모선을 효과적으로 창조한다.

새로운 등록부조종기판은 등록부기억기, 내부클라스터일관성상태기계들과 완충기 그리고 대역적호상접속망의 국부구역을 포함한다.

등록부론리는 내부클라스터호상작용의 밖으로 나가는 부분과 안으로 들어 오는 부분을 위해 쓰인 론리선을 따라 두개의 론리판사이에서 분리된다.

그물망들은 확대가능한 국부기억기와 전역기억기의 대역너비를 지원하며 국부성을 보상한다.

일관성캐쉬들을 가진 단일주소공간은 콤파일러들과 프로그람작성자들이 응용성능조절을 쉽게 허용하고 시공간국부성을 쉽게 개발하게 한다.

성능개선에 기여하는 다른 인자들로는 원격기억기접근지연시간을 은페시키는 완화형기억기일치성방식과 자료미리꺼내기기구를 들수 있다.

Dash계층기억기 Dash에서 기억기공유는 캐쉬행준위에서 진행된다. Dash분산체계등록부를 쓴 쓰기-무효일관성규약을 가지고 있다.

기억기에서 캐쉬행 혹은 국부캐쉬에서 캐쉬복사는 다음의 3상태중의 어느 하나로 될수 있다.

ㄱ) Dash구성방식

ㄴ) 분산기억기계층

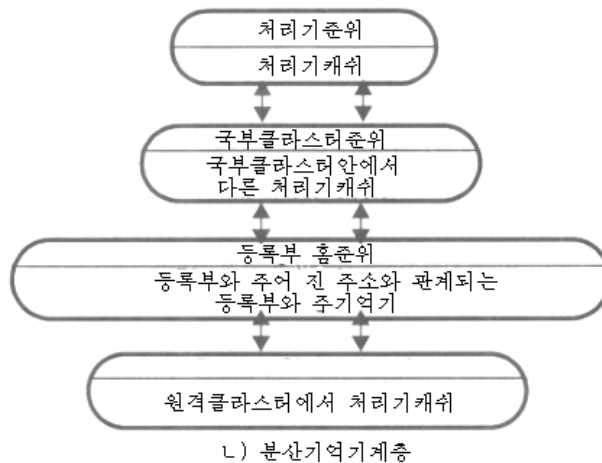
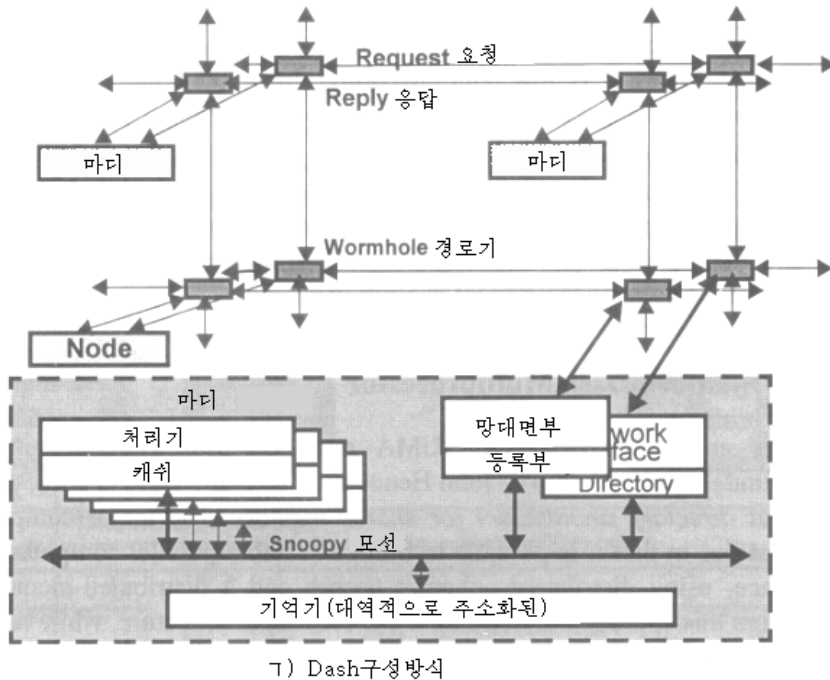



그림 5-14. Stanford Dash CC-NUMA 다중처리기개발계획

- 비캐쉬 : 어떤 마디클라스터에서 캐쉬되지 않는다.
- 공유 : 하나 혹은 그이상의 마디클라스터들의 캐쉬에서 변경되지 않는 상태
- 오물(dirty) : 어떤 마디클라스터의 단일캐쉬에서 변경된다. 등록부는 매 캐쉬행에 대한 요약정보를 보관하고 캐쉬하는 상태와 마디클라스터들을 지정한다.

Dash기억기체계는 그림 5-14 ㄴ)에서 보여 주는바와 같이 론리적으로 4개의 준위계층으로 나눈다.

첫번째 준위는 매 개별적인 캐쉬가 처리기속도와 조화되고 국부모선에서 Snooping을 지원하도록 설계되였다. 처리기가 캐쉬에 접근하는데는 한박자만 걸린다.

처리기캐쉬에 의해 봉사될수 있는 요청은 국부마디클라스터안에 있는 캐쉬들로 보낸다.

Dash원형은 국부마디에서 캐쉬들에 접근하는데 30처리기박자를 허용한다. 이 준위는 요청하는 처리기의 마디령역안에서 다른 모든 처리기들의 캐쉬들을 포함한다. 그렇지 않으면 요청은 홈클라스터준위로 보낸다.

홈준위는 주어 진 기억기주소에 대한 등록부와 물리적기억기를 포함하는 마디클라스터로 이루어 진다. 그것은 홈준위에서 등록부에 접근하는데 100처리기박자가 걸린다. 많은 기억기접근에 대하여(실례로 전용자료참조) 국부클라스터와 홈클라스터는 같으며 계층은 3개 준위로 된다.

일반적으로 요청은 그물망에서 홈클라스터로 움직인다. 홈마디클라스터는 보통 요청을 즉시에 만족시킬수 있지만 등록부입력자료가 오물상태에 있거나 요청처리기가 파도한 접근을 요청할 때에 공유상태에 있으면 네번째 준위가 접근된다. 캐쉬행을 위한 원격클라스터는 캐쉬행복사를 가짐으로써 등록부에 의해 표식된 마디클라스터들로 이루어 진다. 그물망이 련결된 Dash견본설계의 원격마디클라스터에서 처리기캐쉬에 접근하는데 135처리기박자가 걸린다.

5. 4. 4. Dash 에서 등록부에 기초한 규약

등록부기억기는 기억기요청에 대한 조사를 통하여 처리기캐쉬들을 해제한다. 홈마디에는 매 캐쉬행프렘을 위한 등록부입력자료가 있다. 매 입구는 처리기캐쉬당 한개의 존재비트를 포함한다. 추가적으로 상태비트는 기준이 다중캐쉬들에서 비캐쉬되거나 공유되며 한개 캐쉬에 의해 배타적으로 취하게 되는것을 가리킨다.

상태와 존재비트를 리용하면 기준이 씌여 질 때 어느 캐쉬들이 무효가 되여야 하는가를 기억기는 알수 있다. 마찬가지로 등록부는 어느 기억기의 캐쉬행복사가 지금까지의 것인가 혹은 어느 캐쉬가 가장 최근의것을 유지하고 있는가를 가리킨다.

기억기와 등록부가 독립단으로 갈라 지고 확대가능한 호상접속에 의해 처리기에 련결된다면 기억기체계는 확대가능한 기억기대역너비를 제공할수 있다.

등록부기억기를 리용함으로써 장소에 쓰는 마디는 점대점부호나 갱신된 통보문들을 그 기준에 캐쉬한 처리기들에 보낼수 있다. 이것은 snoopy규약에서 요구되는 무효방송과 대조적이다. Dash의 확대가능성은 방송을 피하는 이 능력에 달려 있다.

등록부에 기초한 규약의 다른 중요한 의의는 어떤 특정한 호상작용망기술에 의거하지 않는다는것이다. 결과 그물망 혹은 하이퍼립방체와 같은 통보문넘기기다중콤퓨터를 위해 개발된 낮은 지연시간을 가진 확대가능한 망들을 쉽게 리용할수 있다.

실례 5.8. Dash다중처리기에서 캐쉬등록부규약

그림 5-15 ㄱ)에 오물상태에 있는 등록부입력자료를 가진 원격기억기에로의 읽기요청흐름을 제시한다. 읽기요청은 전용오물마디에 대하여 진행한다. 전용마디는 두개의 응

답통보문을 읽기로 보낸다. 자료를 포함하는 통보문은 직접 요청클라스터로 보내며 공유후쓰기요청은 홈클라스터로 보낸다.

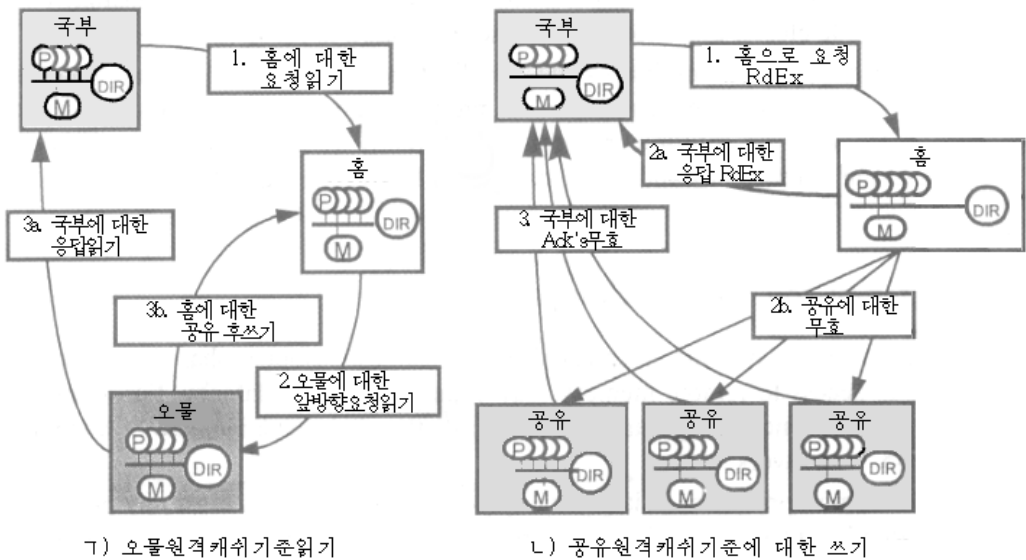



그림 5-15. Dash에서 등록부에 기초한 캐쉬일관성규약

공유후쓰기요청은 캐쉬행을 기억기의 뒤에 쓰고 등록부를 갱신한다.

이 규약은 오물마디가 요청마디클라스터에 직접 응답하도록 함으로써 지연시간을 감소시킨다. 그밖에 이 앞방향전략은 등록부조종기가 모든 요청들의 상태를 유지해야 하는 복잡성이 없이 많은 요청들(즉 다중스레드화된)을 동시에 처리하게 한다.

직렬화는 단일내부클라스터모선호상작용의 시간으로 감소된다.

내부클라스터통보문들을 보내는 동안 보관된 자원만이 출발마디의 원격접근캐쉬에 있는 단일입력자료이다.

그림 5-15 ㄴ)에서 원격봉사를 요구하는 쓰기동작에 대한 대응하는 순서를 보여 준다. 쓰기무효규약은 쓰기를 완료하기전에 캐쉬행의 배타적인 소유권을 얻는 처리기(사실상 쓰기완충기)를 요구한다.

따라서 쓰기가 비캐쉬화된 기준으로 되거나 공유상태에서만 캐쉬된다면 처리기는 국부모선에서 읽기-배타적인 요청을 발행한다.

이 경우 다른 캐쉬는 국부마디클라스터에서 기준입구오물을 가지지 않는다. 그래서 RdEx요청(통보문1)은 홈클라스터로 보낸다. 사전에 원격접근캐쉬입력자료는 국부클라스터에 배당된다.

홈클라스터에서 pseudo-CPU는 읽기-배타적요청을 모선에 발행한다. 등록부는 기준이 공유상태에 있다는것을 가리킨다. 이것은 등록부조종기가 RdEx응답(통보문 2a)을 국부클라스터로, 무효요청(Inv-Req, 통보문 2b)을 공유클라스터로 보내기때문이다.

홈마디클라스터는 캐쉬행을 가지고 있다. 그래서 등록부를 오물상태로 즉시에 갱신할수 있으며 따라서 국부마디는 캐쉬행의 배타적복사를 가지고 있다.

RdEx 응답통보문은 응답조종기에 의해 국부마디에서 받게 되며 그다음 읽기-배타적 요청을 만족시킬수 있다.

해제점에서 일치성을 담보하기 위하여 원격접근캐쉬입력자료는 무효지식의 수(초기 응답통보문에서 보낸 무효 총수와 등가이다.)(InVAcK, 통보문 3)를 받을 때에만 배정되지 않는다.

등록부에 기초한 규약의 중요한 특징은 그것의 앞방향처리방법이다.

마디가 주어 진 요청에 대한 등록부에 응답할수 없으면 요청의 응답가능성을 응답할수 있는 마디클라스터로 전진시킨다. 이 기술은 또한 내부마디통보문이 전진하는 동안은 차단되지 않기때문에 요청의 직결화를 최소화하게 한다.

앞방향처리방법은 등록부조종기가 다중요청들을 앞의 요청들에 대한 추가적인 상태를 보존함이 없이 병행으로 처리하도록 한다.

5. 5. 지연시간허용기술

미래의 확대가능한 체계는 대체로 분산공유기억기구성방식을 리용할것이다. 원격기억기에 대한 접근은 긴 지연시간을 동반한다. 더우기 처리기속도는 기억기와 호상접속망의 속도증가보다 훨씬 빨리 증가하고 있다.

확대가능한 다중처리기나 혹은 대규모다중콤퓨터클라스터들은 지연시간감소, 지연시간회피 그리고 지연시간은페기구들의 리용에 의거하여야 한다.

4개의 지연시간은페기구들이 미래의 체계들의 확대가능성과 프로그람화가능성을 높이기 위하여 연구고찰되고 있다.

5. 5. 1. 지연시간회피, 감소, 은페

기억기지연시간문제를 풀기 위한 3가지 방법들은 지난 시기에 제기되였다. 간단한 실례를 들어 이 방법들을 소개한다.

지연시간회피 이 기술은 자료/프로그람국부성을 얻기 위하여 사용자응용을 구성방식, 콤파일러 혹은 응용준위에서 조직하려고 시도한다.

그 목적은 원격자료접근 혹은 프로그람접근에서 긴 지연시간을 피하는것이다. 이것은 응용프로그람들이 시간국부성 혹은 공간국부성을 공개할 때만 가능하다. 시공간국부성을 높이기 위한 많은 기술이 개발되였다.

이 기술들을 다음의 3가지로 분류할수 있다.

- **구성방식지원** 구성방식지원은 여러가지 형태의 캐쉬일관성규약, 기억기일치성모형, 고속통보문넘기기 그리고 확대가능한 체계에서 제작될수 있는 동기화하드웨어를 포함한다.
- **사용자지원** 프로그람작성자(사용자)는 국부성을 높이기 위하여 응용프로그람을 명확히 쓸수 있다. 고성능 Fortran(HPF)과 같은 어떤 언어들은 유연성을 가지고

사용자들이 국부성을 조사하는것을 도와 준다.

방법은 국부접근들을 보장하기 위하여 자료모임을 얼마나 잘 배당하는가를 콤파일러에게 명령하는것이다. 이것을 실례 5.9에서 보여 준다. 자세한것은 14.3.2에서 토론한다.

쏘프트웨어지원 콤파일러와 같은 체계쏘프트웨어는 국부성을 높이기 위한 일정한 변환을 수행할수 있다. 실례 5.10에서 고찰된다. 자세한것은 12.3.5에서 토론한다.

실례 5.9. 분산자료구조의 국부성

복소수들의 배렬은 과학계산과 신호처리응용에서 자주 쓰인다. 두가지 일반적인 방법은 다음과 같다.

```
/***** Method(1) Array of Structure*******/
typedef struct{double real,image;}CUMPLEX;
COMPLEX data[N1][N2][N3];
/***** Method(2) Separate Arrays********/
double data_real[N1][N2][N3], data_image[N1][N2][N3];
```

이 두 방법들은 같은 알고리듬문제를 푸는데 쓰일 때 성능차이를 잘 보여 준다.

방법 1은 대부분의 초고속계산응용에 쓴다. 왜냐하면 방법 1은 더 좋은 국부성을 가지기때문이며 따라서 보다 작은 캐쉬비적중률에 귀착된다.

실례 5.10. 고리변환(loop)에 의한 자료국부성에서의 실행

행렬곱하기 C=A×B를 수행하는 전통적인 알고리듬은 아래와 같다.

```
for(i=0;i<N;i++)
   for(j=0;j<N;j++)
      for(k=0;k<N;k++)
         c[i][j]+=a[i][k]*b[k][j];
```

고리교환(interchange)이라고 부르는 변환은 더 큰 자료국부성을 개발하는 콤파일러에 의해 수행될수 있다.

아래에서 보여 주는바와 같이 k고리와 j고리는 서로 교환된다.

이 변환은 계산된 결과를 변화시키지는 않지만 크기차수이상으로 성능을 향상시킨다.

```
for(i=0;i<N;i++)
   for(k=0;k<N;k++)
      for(j=0;j<n;j++)
```

```
c[i][j]=a[i][k]*b[k][j];
```

지연시간감소 Burton Smith[569]가 강조한바와 같이 자료국부성은 제한될수 있고 발견하기 어려우며 동적으로 변할수 있다.

대부분의 정렬(sorting)알고리듬들은 모두 3가지 특징을 가지고 있다.

국부성이 지연시간을 완전히 피할수 없을 때 원격참조에 의해 생기는 긴 지연시간을 감소하는 기술이 필요하다.

확대가능한 콤퓨터의 기본설계목표는 통신부분체계를 효과적으로 만드는것이다. 그러자면 다음의 성능이 필요하다.

- 효과적인 호상접속하드웨어(6장에서 취급한다.)
- 효과적인 망대면부(6장에서 취급한다.)
- 고속통신쏘프트웨어(7장과 14장에서 취급한다.)

지연시간은폐 이 기술은 계산에서 통신지연시간을 은폐시키는것을 가리킨다. 즉 어떤 중복기술들을 통하여 지연시간은폐를 다음의 4가지 방법으로 완성할수 있다.

- **미리꺼내기기술** 이 기술은 명령이나 자료가 실지로 참조되기전에 그것을 처리기가까이로 날라 간다. 다른 말로 말하면 자료미리꺼내기는 읽기지연시간을 은페시킨다.
- **분산일관성캐쉬** 하드웨어에 의해 지원된 분산일관성캐쉬들은 캐쉬비적중들을 감소하거나 clean복사를 회복시키는 시간을 짧게 한다.
- **완화형기억기일치성모형** 기억기참조동작의 완충화와 관흐름화를 리용한다. 완화형기억기일치성을 리용하는것은 쓰기지연시간을 은페시킨다는것을 의미한다.
- **다중문맥처리기** 처리기가 긴 지연시간동작을 만났을 때 한 문맥에서 다른 문맥에로 절환하도록 한다.

5.6에서 하드웨어지원에 의한 다중스레드화를 고찰한다.

가장 발전된 회피나 감소기술일지라도 모든 응용프로그람을 위한 대규모병렬체계에서 지연시간을 제거할수 없다.

지연시간은페기술들은 긴 지연시간이 존재하는 속에서 고성능을 얻는데 목적을 두고 있다. 지연시간을 허용하는 기본방법은 처리기가 자료접근이 진행중에 있는 동안 쓸모있는 계산을 계속 해나가도록 하는것이다. 즉 중복계산과 통신.

지연은페기술의 3분의 1은 이 절에서 제시한다. 다중스레드화방법은 5.6절에서 취급한다.

5. 5. 2. 분산일관성캐쉬

일관성을 가진 분산캐쉬들을 리용하는것은 국부캐쉬비적중을 줄일뿐아니라 캐쉬행준위에서 DSM을 얻는 효과적인 방도를 제공한다. 이전 절들에서 본바와 같이 하드웨어와 쏘프트웨어실행은 분산일관성캐쉬들을 지원할수 있다.

아래에서는 분산일관성캐쉬들을 가질 때 성능증가에서의 효과성을 보여 주는 몇가지 성능평가기준결과들을 보기로 한다.

일관성캐쉬개발과 제품들 MIT Alewife, KSR−1과 Stanford Dash는 모두 실행형등록부에 기초한 일관성규약들을 가지고 있다. 분산캐쉬는 동기화−적재문제가 아니라 원격−적재문제의 풀이를 제공한다.

다중스레드화는 원격적재와 동기화적재문제의 풀이를 제공한다. 그러나 두 방법들을 두가지 형태의 원격−접근문제를 풀기 위해 결합할수 있다.

캐쉬일관성문제는 Snoopy일관성규약을 리용하여 모선에 기초한 작은 규모의 다중처리기들에 대해 쉽게 풀수 있는데 점대점 혹은 절환형호상접속을 리용하는 대규모다중처리기들에 대해서는 문제가 훨씬 복잡하다. 결과 현재의 대규모다중처리기들은 캐쉬들에 대한 하드웨어지원을 제공하지 않는다(실례로 BBN Butterfly 혹은 Cray T3D/T3E).

다른것들은 쏘프트웨어에 의해 일관성을 유지하는 캐쉬들을 제공한다(실례로 IBM RP3).

4개의 최근 NUMA다중처리기체계들을 보면 SGI Origin2000, HP/Convex Exampar X Class, Sequent NUMA−Q2000과 Data General NUMA봉사기들을 들수 있는데 모두 CC−NUMA개발경험들로부터 연구제작되였다. 그것들은 모두 등록부에 기초한 캐쉬일관성규약으로 실행된다.

성능평가기준조건들 여기서는 Dash하드웨어일관성캐쉬들에서 허용된바와 같이 전용자료와 공유자료가 캐쉬가능할 때 리득을 평가하며 전용자료만이 캐쉬가능한 경우와 비교한다. 그림 5−16에서는 3개 응용프로그람의 매 경우에 대하여 공유자료의 캐쉬를 가질때와 가지지 않을 때 정규실행시간분석을 보여 준다.

전용자료는 두가지 형태의 캐쉬들에 캐쉬된다.

매 성능평가기준에 대한 왼쪽 막대기는 캐쉬가 없는 경우에 대응하고 오른쪽 막대기는 공유 읽기/쓰기자료의 캐쉬효과를 보여 준다. 3개의 평가기준프로그람들을 보면

MP3D : 이것은 항공술연구에서 쓰인 립자에 기초한 3차원모의기이다.

LU : 이것은 대규모선형방정식계를 푸는 분해프로그람이다.

PTHOR : 이것은 수자론리모의프로그람이다.

이 성능평가기준프로그람들은 Stanford Dash다중처리기에서 검사되였다. 여기서는 매 형태의 지연시간은폐기구에 대한 성능결과들이 분석된다. 매 응용프로그람의 실행시간은 공유자료가 캐쉬되지 않는 경우에 대해 정규화된다.

매 막대기의 바닥부분의 수자는 동작시간 혹은 처리기에 의해 실행된 주기들로 표현

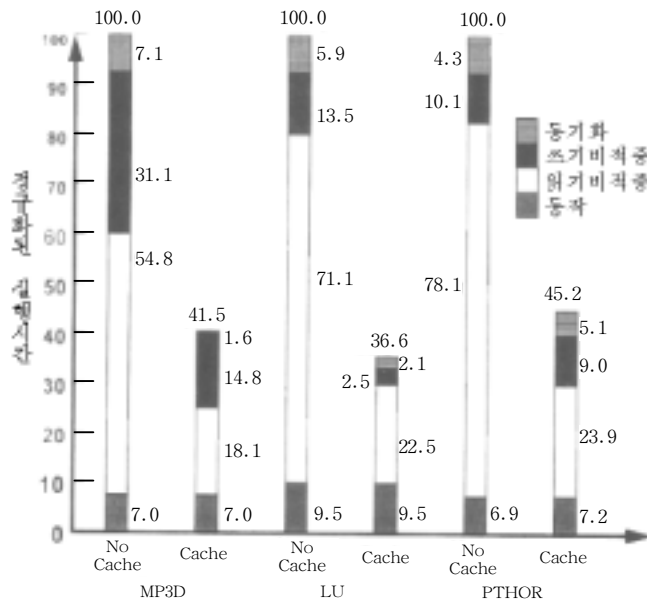



그림 5-16. Dash평가기준실험에서 모의된 공유자료에 대한 캐쉬효과

하고 그우에 있는 부분의 수자는 처리가 읽기를 완성하는데 기다리는 시간을 표현하며 또 그우에 있는 부분의 수자는 처리기가 쓰기를 완성하는데 기다리는 시간을 표현한다. 꼭대기부분 수자는 동기화를 반영하는것으로써 처리기가 차단(lock)과 장벽으로 인하여 멎는 시간을 표현한다.

캐쉬의 리득 기대한바와 같이 공유읽기/쓰기자료의 캐쉬는 3개의 Stanford성능평가기준 프로그람에서 2.2~2.7배의 실질적인 성능리득을 보여 주었다.

가장 큰 리득은 읽기비적중으로 소비된 주기수감소에서부터 온다. 리득의 크기는 각이한 쓰기적중률로 인하여 3개 프로그람에서 변하지만 쓰기비적중으로 소비된 주기들은 역시 감소된다.

MP3D, LU 그리고 PTHOR에 의해 도달캐쉬적중률은 공유읽기참조에 대하여 각각 80%, 66%, 77%이고 공유쓰기참조에 대해서는 각각 75%, 97%, 47%이다.

이 적중률은 보통 단일처리기적중률보다 실질적으로 낮다.

적중률이 낮은것은 여러가지 인자에 기인된다.

공학응용을 위한 자료모임의 크기는 크고 병렬성은 이 응용프로그람들에서 공간국부성을 감소시키며 처리기들사이의 통신은 무효비적중에 귀착된다. 아직 하드웨어캐쉬일관성성능은 성능을 실질적으로 높이는 효과적인 기술이다. 이 증가는 콤퓨터조작자나 프로그람작성자의 도움이 없이 이루어 진다.

5. 5. 3. 자료미리꺼내기방법

자료미리꺼내기는 관련 있는 자료들이 참조되기전에 처리기로 접근시키는 프로그람에서 예상되는 비적중에 대한 지식을 리용하여 처리기근방으로 이동한다.

미리꺼내기는 하드웨어 혹은 쏘프트웨어의 두가지에 의해 조종될수 있다.

미리꺼내기형태들 미리꺼내기는 속박형과 비속박형으로 분류할수 있다. 속박형미리꺼내기에 대해서 보면 후에 참조하는 값이 미리꺼내기가 완성될 때 작업등록기들에 직접 속박되거나 적재된다.

이것은 미리꺼내기와 실지참조들사이동안에 다른 처리기가 좋은 위치로 변경시키면 값이 무효로 되므로 속박형미리꺼내기가 발행될수 있을 때에 제한을 준다. 속박형미리꺼내기는 성능저하를 가져 올수 있다. 반대로 비속박형미리꺼내기는 자료를 캐쉬로만 가져 가며 캐쉬일관성규약을 지키면서 처리기가 실지 값을 읽을 때까지 일치성을 유지한다. 하드웨어조종에 의한 미리꺼내기는 긴 캐쉬행들과 명령예측과 같은 도식들을 포함한다.

긴 캐쉬행들의 효률성은 다중처리기응용들에서 감소된 공간국부성에 의해 제한되며 명령예측은 분기들과 유한예측완충기크기에 의해 제한된다.

쏘프트웨어조종에 의한 미리꺼내기에서는 명백한 미리꺼내기명령들이 사용자코드에서 발행된다. 쏘프트웨어조종은 미리꺼내기가 선택적으로 진행되게 하고(따라서 대역너비요구를 감소시킨다.) 미리꺼내기발행과 실지참조사이간격을 가능한껏 확장하는데 이것은 지연시간들이 커질 때 아주 중요하다.

쏘프트웨어조종의 결점들을 보면 정교한 쏘프트웨어개입에 대한 요구는 물론 미리꺼내기를 생성하는데 요구되는 외부명령부가처리를 포함할수 있다. 여기서는 비속박쏘프트웨어조종에 의한 미리꺼내기에 중심을 둔다.

미리꺼내기의 리익 그 리익은 몇가지 원인에서 볼수 있다. 가장 명백한 리익은 기준이 참조되는 시간까지 캐쉬에 있는 코드에서 미리꺼내기가 충분히 빨리 발행될 때 생긴다. 그러나 미리꺼내기는 이것이 가능하지 않을 때에도(즉 자료구조주소가 그것이 참조되기전까지 즉시에 결정될수 없을 때) 성능을 개선할수 있다.

다중미리꺼내기가 자료구조를 꺼내기 위해 뒤에서 뒤로 발행된다면 먼저 미리 꺼낸 참조들을 제외한 모든 지연시간은 기억기접근의 관흐름으로 인해 은폐될수 있다.

미리꺼내기는 소유권에 기초한 캐쉬일관성규약을 리용하는 다중처리기들에서 또 다른 리득을 준다.

캐쉬행이 변경되면 소유권을 가지고 그것을 직접 미리꺼내기하는것이 소유권을 얻기 위한 쓰기지연시간들과 안전한 망통행을 뚜렷하게 감소시킬수 있다. 망통행은 소유권을 가진 미리꺼내기가 읽기-공유복사를 먼저꺼내기하는것을 피하기때문에 읽기-변경-쓰기명령들에 의해 감소된다.

Stanford 성능평가기준평가들 Stanford연구자들(Gupta, Hennessy, Gharacherloo, Mowry와 Weber, 1991)은 여러가지 지연시간은페기구들에 대한 성능평가기준결과들을 보고하였다.

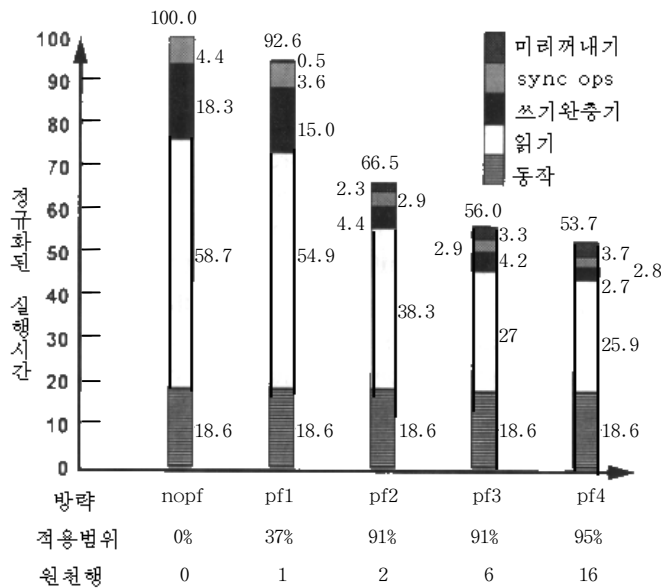



그림 5-17. Dash 다중처리기의 MP3D 에서 여러가지 미리꺼내기방법들의 효과

미리꺼내기효과를 그림 5-17에서 보여 주는데 Stanford Dash다중처리기의 MP3D에 기초하였다.

모의실행은 5개 단계에서 64×8×8공간배렬의 10,000립자들을 포함한다.

5개의 미리꺼내기방법들이 검사되였다(그림 5-17에서 *nopf*, *pf*1, *pf*2, *pf*3, *pf*4).

이 방법들은 미리꺼내기가 없는 *nopf*에서부터 시작하여 같은 반복에서의 립자기록 혹은 반복증가수에 따라 관흐름화된것(즉 *pf*1에서 *pf*4)의 미리꺼내기로 배렬한다.

막대기들은 *nopf* 방법에 관해 정규화된 실행시간을 보여 준다.

매 막대기는 미리꺼내기, 동기화동작들을 위해 요구되는 시간분석을 보여 주는데 쓰기완충기, 읽기 그리고 동작계산(busy)을 리용한다.

마지막 결과는 미리꺼내기들이 미리꺼내기가 없는 경우에 생기는 95%까지의 비적중들에 대해 발행된다는것이다(그림 5-17에서 적용범위(coverage)인자로 표현된다.).

미리꺼내기는 쓰기완충기읽기동작실현을 리용하여 동기화동작들에서 뚜렷한 시간감소를 일으킨다.

*Pf*4미리꺼내기방법을 *nopf*전략과 비교할 때 도달된 가장 좋은 속도증가는 1.86이다. 아직 미리꺼내기의 리득은 응용의존성을 가진다. MP3D코드에 미리꺼내기를 도입하기 위하여 수천개의 명령행들을 포함하는 원천코드에 16개의 외부코드행만을 추가하였다.

5. 5. 4. 완화형기억기일치성효과

그림 5-18에 3개의 성능평가기준응용에서 SC(순차일치성)와 RC(해제일치성)의 실행시간분석을 보여 준다.

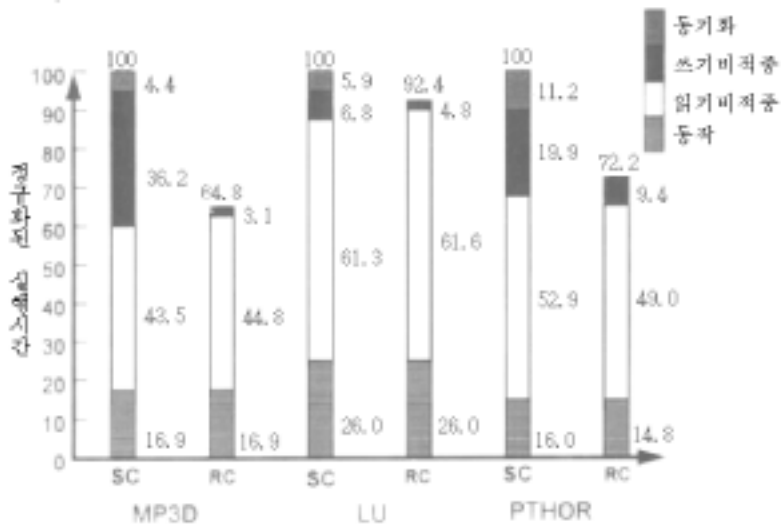



그림 5-18. 순차일치성(SC)과 해제일치성(RC)에서 공유기억기모형을 완화시키는 효과

실행시간은 그림 5-18에서 보여 준 캐쉬된 공유자료들에 관해 정규화된다. 결과들로부터 볼수 있는바와 같이 RC기억기모형은 쓰기-비적중지연시간으로 인한 모든 휴지(idle)시간을 제거한다.

MP3D와 PTHOR의 SC기억기모형에서 쓰기-비적중시간은 큰 비률을 이루고 있기때문에(각각 35%, 20%) MP3D와 PTHOR에서 성능리득은 크다. 또한 LU의 SC기억기모형에서 쓰기-비적중시간은 상대적으로 작기때문에(7%) LU에서 성능리득은 작다.

5. 6. 다중스레드화된 지연시간은폐

다중스레드화는 문맥절환기초우에서 다중문맥을 동시에 조종할수 있도록 처리기가 설계될것을 요구한다.

이 경우 처리기는 다른 프로그람스레드를 실행하기 위하여 절환되며 현재 스레드는 자료/명령접근이 완성될 때까지 대기한다. 여기서는 다중문맥처리기들의 구성방식을 개괄한다. 다중스레드화의 효과와 처리기효률성의 개선을 고찰한다.

5. 6. 1. 다중스레드화된 처리기모형

확대가능한 병렬콤퓨터는 처리기(P)들의 망과 기억기(M)마디들로 모형화된다(그림 5-19 ㄱ). 분산기억기들은 전역주소공간을 이룬다. 다음의 4개 기계파라메터들은 체계성능해석을 위해 정의된다.

- **지연시간 *L*** 이것은 원격기억기접근에 대한 경험으로 얻은 통신지연시간이다. *L* 값은 망지연, 캐쉬-비적중벌칙과 분할처리에서 충돌로 인한 지연을 포함한다.
- **스레드수 *N*** 이것은 처리기에 끼워 넣을수 있는 스레드수이다. 스레드는 프로그람계수기, 등록기모임과 요구되는 문맥상태단어들로 이루어 진 문맥에 의해 표현된다.
- **문맥-절환-부가처리 *C*** 이것은 처리기가 문맥절환을 수행할 때 잃은 주기들을 가리킨다. 이 시간은 절환기구와 활성화된 스레드들을 유지하는데 요구되는 처리기상태들의 수에 의거한다.
- **문맥절환들사이구간 *R*** 이것은 원격참조에 의해 기동된 문맥절환들사이의 주기들로 표현되는 동작길이를 가리킨다.

p=1/R를 원격접근을 위한 **요청률**이라고 부르는데 프로그람동작과 기억기체계설계의 결합을 반영한다.

처리기효률성을 높이기 위한 한가지 방법은 분산일관성캐쉬들을 리용하여 요청률을 감소시키는것이다. 다른 방법은 다중스레드화를 통하여 처리기대기를 없애는것이다.

다중스레드화의 기본개념을 아래에서 서술한다.

다중스레드화된 계산들 Bell[67]은 그림 5-19 ㄴ)에서 보여 준 다중스레드화된 병렬계산모형의 구조를 서술하였다. 계산은 순차스레드(1)에서 출발하여 처리기들이 다중스레드들의 계산(3)을 시작하는 감독일정짜기(2), 콤퓨터가 분산기억기(4)를 가질 때 마디들에서 변수들을 갱신하는 중간콤퓨터통보문들, 그리고 마지막으로 병렬작업(5)의 다음단시작이전의 동기화에 따라 진행한다.

분산기억기구조에서 고유한 통신부가처리기간은 보통 계산과정에 분포되며 중복될수 있다.

다중콤퓨터들에서 통보문들과 부가처리(보내기와 받기호출)는 병렬계산으로 동작하는 전용하드웨어에 의해 감소될수 있다.

계산 grain을 완성하기 위하여서는 일정한 량의 자료들을 다른 매듭으로 이동하여야 하므로 통신대역너비는 립도를 제한한다. 통보문넘기기호출(4)과 동기화(5)는 비생산적이다. 이 지연들을 감소시키거나 은페시키는 고속기구들이 요구된다. 다중스레드화는 단일스레드실행에서 속도를 증가시킬수 없지만 약한 순서화 혹은 완화형일치성모형들은 속도를 증가시킬수 있다.

다중스레드화된 체계들은 다중문맥(혹은 다중스레드화된)처리기들로 구성된다. 이 절에서는 SaaVedra 등의 문헌에 기초한 추상모형을 고찰한다[521].

기억기지연시간(L), 문맥수(N) 그리고 문맥절환부가처리(C)의 함수로서 처리기효률성

발행을 고찰한다.

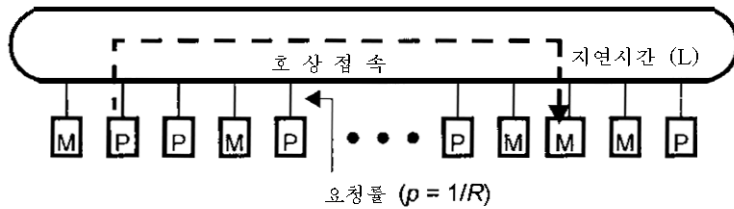
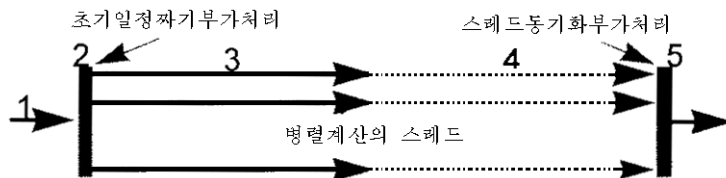
처리기효률성 단일스레드처리기는 원격참조가 지연시간 L주기를 가지는 동안 대기한다. 그림 5-20에서 모형화된바와 같이 다중스레드처리기는 현재문맥을 중지하고 다른 문맥에로 절환한다.

ㄱ) 확대가능한 병렬체계의 추상모형

ㄴ) MPP체계에서 다중스레드화의 개념

그림 5-19. 확대가능한 다중처리기체계를 위한 다중스레드계산모형

원격참조가 결정되지 않아도 문맥절환부가처리의 몇주기만 해보고 처리기는 다시 동작하여 유용한 작업을 해나간다. 모든 유효한 문맥들이 차단되기만 하면 처리기는 휴지상태에 있게 된다.

명백한 목적은 처리기가 동작하는 시간을 최대화하는것이다. 다중스레드처리기의 효률성을 다음과 같이 정의한다.

$$\text{효률성} = \text{동작} / (\text{동작} + \text{절환} + \text{휴지}) \qquad (5.12)$$

여기서 동작, 절환 그리고 휴지는 처리기가 대응하는 상태에 있을 때의 시간량을 표시한다. 다중스레드처리기의 기본방법은 절환시간을 증가시키지 않고 부가처리를 감소시키기 위해 몇개 문맥들의 실행을 끼워 넣는것이다.

처리기상태는 처리기에서 여러가지 문맥들의 배치에 달려 있다. 그것의 수명시간동안에 문맥주기는 다음의 상태들을 통과한다. 즉 준비, 실행, 출발 그리고 차단.

기껏해서 한개 문맥실행 혹은 출발이 있을수 있다.

문맥이 실행상태에 있으면 처리기는 동작하고 한개 문맥에서 다른 문맥으로 이행하면 즉 문맥이 출발할 때 처리기는 절환한다.

만일 그렇지 않으면 모든 문맥들이 차단되며 처리기는 부가처리상태에 있다고 본다.

문맥실행은 문맥절환을 요구하는 동작을 발행할 때까지 처리기가 동작상태에 있게 한다.

문맥은 출발상태에서 C주기를 소비하고 L주기동안 차단상태에 들어 가며 마지막에

읽기상태로 다시 들어 간다.

추상처리기모형 다중스레드처리기를 그림 5-20에서 모형화하였다. 간단히 하기 위해 문맥당 한개 스레드를 가정하고 매 문맥은 자기의 하드웨어자원들로 표시된다고 가정한다.

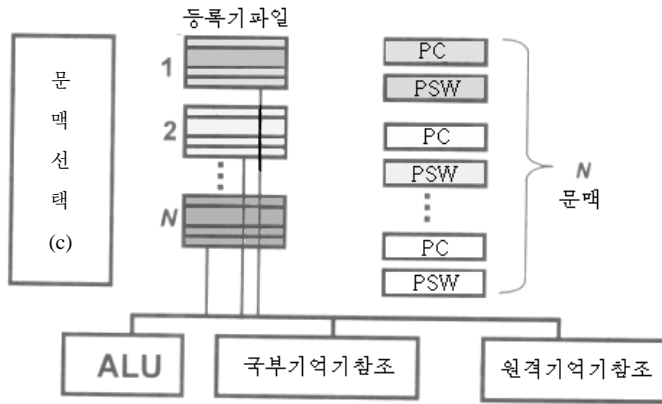



그림 5-20. 문맥당 한개 스레드를 가진 다중문맥처리기모형

즉 프로그람계수기, 등록기파일 그리고 주소상태단(PSW)이다.

매 문맥은 이 자원들로 충분히 실행가능하다고 하자. 다중등록기파일들을 가지고 한 문맥에서 다른 문맥에로의 문맥-절환부가처리를 한개 혹은 두개 처리기주기들만큼 짧게 할수 있다.

5. 6. 2. 문맥절환방책

각이한 스레드화된 구성방식들은 리용된 문맥-절환방법들에 의해 구별된다. 4가지 절환방법들을 아래에서 지정한다.

(1) **캐쉬비적중에서 절환** 문맥은 캐쉬비적중을 만날 때 절환된다. 이 경우 R는 캐쉬비적중들사이의 평균구간(주기들에서)이고 L은 캐쉬비적중에서 다시 회복하는데 요구되는 시간이다.

(2) **모든 적재에서 절환** 이 방법은 모든 적재동작에서 절환을 하게 하는데 비적중과는 독립이다. 이 경우 R는 적재들사이 평균구간을 표시한다. 일반모형은 절환다음에 L주기동안 문맥이 차단된다는것을 가정한다. 그러나 이것은 절환-on-적재처리기인 경우 적재가 캐쉬비적중을 일으킬 때에만 있을수 있다. 두개의 지연시간(L_1과 L_2)이 있어서 매개는 모든 절환에서 발생할 확률(p_1와 p_2)을 가진다고 가정하면 일반모형을 쓸수 있다. L_1이 캐쉬비적중에서의 지연시간을 표현하면 p_1은 비적중률에 대응한다. 지연시간 L_2은 확률 p_2을 가진 령주기기억기지연시간이다.

(3) **모든 명령에서 절환** 이 방법은 적재와는 독립인 모든 명령에서 절환하도록 한다. 다른 말로 주기-by-주기기초우에서 각이한 스레드들을 끼워 넣는다. 련속적인 명령들은 독립으로 실행되며 관흐름화된 실행으로 리득을 얻는다. 그러나 캐쉬

비적중은 국부성의 파괴로 늘어 날수 있다. 그것은 문맥의 주기-by-주기엇끼우기가 캐쉬비적중에서의 절환에 비해 성능우점을 보여 주는 추적구동실험에 의해 증명되였다. 이때 문맥엇끼우기는 관흐름의존성을 은페시키거나 문맥-절환비용을 낮출수 있다.

(4) **명령블로크에서 절환** 여러가지 스레드들에 명령블로크들이 끼워 진다. 이것은 국부성보존으로 캐쉬-적중률을 개선한다. 그것 역시 단일문맥성능에 리득을 준다.

단일스레드처리기효률성 단일스레드처리기는 원격참조가 발행될 때까지 R주기동안에 문맥을 실행한다. 그것은 문맥참조가 L주기동안에 완성될 때까지 휴지하게 된다. 문맥절환은 없으며 명백히 절환부가처리도 없다.

이 동작을 주기 $R+L$을 가진 재순환과정으로 모형화할수 있다. R와 L은 각각 처리기동작과 휴지에 대응한다. 따라서 단일스레드처리기효률성은

$$E_1 = \frac{R}{R+L} = \frac{1}{1+L/R} \tag{5.13}$$

으로 정의한다.

이것은 대용량기억기지연시간 L을 가진 병렬체계에서 처리기의 성능저하를 명백히 보여 준다. 그 경우 E_1은 실행길이가 일반적으로 $R \ll L$이기때문에 보통 낮다.

다중스레드처리기효률성 다중문맥과 함께 기억기지연시간은 C주기부가처리를 가진 새로운 문맥으로 절환함으로써 은폐될수 있다.

절환들사이의 실행주기는 문맥수에 따라 변하지 않으며 절환이 생길 때 문맥실행준비가 되여 있다고 가정하자.

그러면 처리기는 부가처리될수 있다. 처리기효률성은 그림 5-21에서 보여 준바와 같이 두개의 서로 다른 조건밑에서 아래와 같이 해석된다.

포화범위 포화범위에서 처리기는 최대리용을 가지고 동작한다. 이 경우 반복동작주기는 $R+C$이고 효률성은 간단히

$$E_{sat} = \frac{R}{R+C} = \frac{1}{1+C/R} \tag{5.14}$$

로 정의된다.

포화에서 효률성은 지연시간 L과 독립이고 문맥수증가에 따라 변하지 않는다. 포화는 단일스레드를 처리하는데 요구되는 시간보다 다른 스레드를 봉사하는데 처리기가 더 많은 시간을 소비할 때 즉 $(N__1)(R+C)>L$일 때 도달된다. 이것은 실행길이상수조건에서 포화점을 준다.

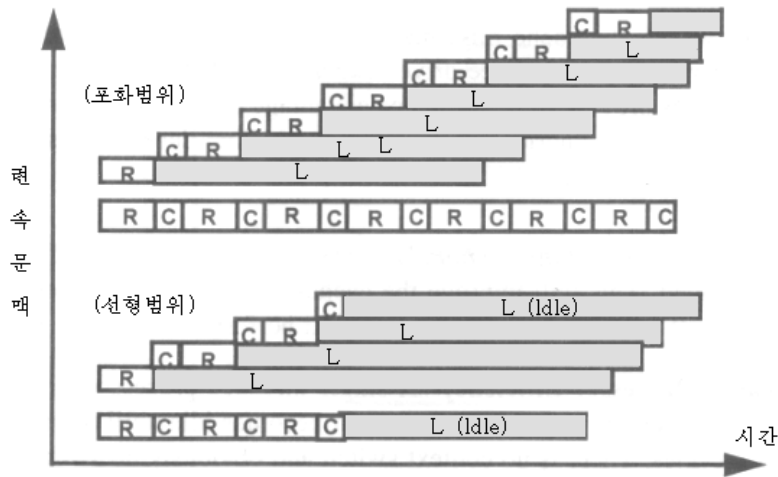



그림 5-21. 포화와 선형범위에서 다중스레드처리기의 문맥절환

즉

$$N_d = \frac{L}{R+C} + 1 \qquad (5.15)$$

선형범위 문맥수가 포화점아래일 때 문맥절환이후에 준비문맥들이 없을수 있다. 그래서 처리기는 일부 휴지주기들을 경험을 통해 얻을수 있다. 준비문맥으로 절환하고 원격참조가 발행될 때까지 그것을 실행하며 참조를 처리하는데 요구되는 시간을 $R+C+L$에 더한다. 이 시간동안에 다른 모든 문맥들은 처리기에서 전환한다. 따라서 효률성은

$$E_{lin} = \frac{NR}{R+C+L} \qquad (5.16)$$

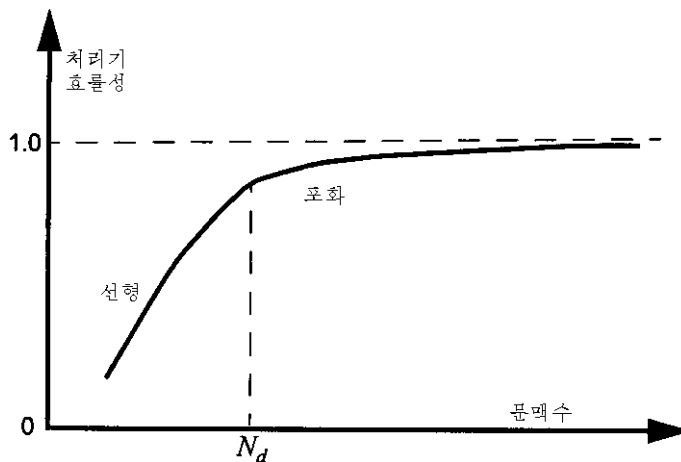



그림 5-22. 문맥수에 따르는 처리기효률성

로 주어 진다.

그림 5−22에서 효률성에 관한 두개 식 (5.14)와 (5.16)은 식 (5.15)의 포화점 N_d에서 만난다. 처리기효률성은 포화점에 이를 때까지 문맥수에 따라 선형으로 증가한다. 그 점을 지나서 E_{sat}곡선은 처리기효률성에 최대한도를 준다.

다중스레드화는 처리기효률성과 망통행을 증가시킨다는것을 알수 있다. 이 두개의 반대되는 목표들사이에 Agarwal[9]에서 토론된바와 같이 절충이 존재한다.

실례 5.11. 다중스레드처리기에서 원격기억기지연시간은페

50~200처리기주기의 원격기억기지연시간을 가진 병렬콤퓨터를 고찰하자. C=1, 4, 16에 대한 문맥−절환부가처리의 3가지 경우를 가정한다.

Saavedra 등 [522]는 스레드실행길이가 R=16 Cycles인 가정밑에서 처리기효률성에 대한 수값결과들을 얻었다. 이 결과들을 표 5−6에 제시한다.

표 5-6　　두개 다중스레드화된 구성방식들에 대한 처리기효률성

문맥절환 부가처리 (C)	N = 2				처리기당 문맥 N = 6			
	L = 50	L =100	L = 150	L = 200	L = 50	L = 100	L = 150	L = 200
1	0.48	0.29	0.22	0.15	0.94	0.76	0.68	0.45
4	0.45	0.27	0.21	0.15	0.80	0.68	0.56	0.44
16	0.36	0.25	0.16	0.14	0.50	0.49	0.45	0.41

여기서 보여 준 결가과들은 파라메터수값들을 식 (5.14), (5.16)에 대입하여 얻었다. 두가지 경우가 평가되였는데 하나는 두개 문맥처리기를 리용한것이고 다른것은 6개문맥처리기를 리용한것이다. 표로부터 다음과 같은 몇가지 흥미 있는 관측결과들을 볼수 있다.

- 두개 문맥처리기를 리용하면 처리기효률은 기억기지연시간의 증가와 높은 문맥절환부가처리에 관해 0.48에서 0.14로 급격히 떨어 진다. 다시 말하여 두 문맥들사이 절환은 50주기이상의 기억기지연시간을 은페시키는데는 충분하지 않다.
- 처리기효률성에서의 저하는 두개 문맥처리기들인 경우보다 6개문맥처리기들에서 훨씬 낮다. 효률성을 두배이상 올리자면 두개 문맥처리기들보다 6개 문맥처리기들을 리용해야 한다.
- 문맥절환의 극히 낮은 부가처리에서(C=1처리기주기) 6개 문맥처리기들을 리용하면 처리기효률성은 0.94와 0.68사이에서 보장하면서 50~150주기의 기억기지연시간을 은페할수 있다.
- C=16주기인 경우 처리기효률성은 0.5이하로 되여 두가지 형태의 처리기구성방식에서 접수할수 없다고 본다.
- 문맥절환부가처리 C=4주기에 대해서 처리기효률성은 기억기지연시간이 증가하는데 따라 C=1주기인 리상적인 경우로 수렴한다.

우의 수값결과에 기초하여 85%효률성을 가진 처리기설계를 추정할수 있다.

문맥절환부가처리는 2주기이상이 아닌 구성방식을 선택한다. 매 처리기에서 문맥수는 75처리기주기까지의 기억기지연시간을 은페시키기 위하여 4로 정한다. Agarwal [9]는 작은 수의 문맥에서 절환만이 체계성능에 리득을 준다는것을 지적하였다. 많은 스레드들은 제한된 자원들의 과도한 문맥절환을 일으켜 경쟁할수 있는데 그러면 성능을 심하게 떨군다. 다중토막화된 기계의 좋은 실례는 Tera다중처리기체계[27]이다. 흥미 있는 실지설계와 성능평가기준실험을 위해 Tera의 개발계획에 종사하려고 할수 있다.

다중스레드기계가 콤퓨터공업에서 상업적가치가 있다는것을 증명하지 못하였다.

5. 6. 3. 지연시간은페결합기구

여러가지 지연시간은페기구를 결합한 효과를 Stanford대학의 MP3D성능평가기준결과에 기초하여 그림 5-23에 보여 준다.

다중문맥(MC)처리기들을 리용한 효과를 PF(미리꺼내기), CC(일관성캐쉬), RC(해제일치성)와 같은 다른 지연시간은페기구와 함께 제시한다.

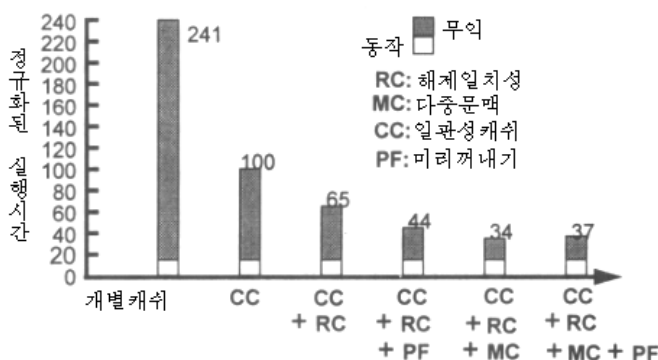



그림 5-23. Stanford Dash 다중처리기의 MP3D 성능평가기준결과에 기초한 여러가지 지연시간은페기술들의 결합효과

실행시간의 동작부분은 모든 기구결합에서 같다. 이것은 MP3D프로그람이 실행하는 CPU동작시간에 대응한다.

막대기에서 휴지부분은 모든 캐쉬비적중벌칙을 포함하는 기억기지연시간에 대응한다.

모든 시간들은 캐쉬일관성체계들에서 요구되는 실행시간(100단위)에 관해 정규화된다.

맨 왼쪽 막대기(241단위)는 공유읽기 혹은 쓰기가 없이 전용캐쉬를 배타적으로 리용하는 가장 나쁜 경우에 대응한다. 긴 부가처리는 이 경우에 과도한 캐쉬비적중에 의해 생긴다. 캐쉬일관성체계의 리용은 전용인 경우에 2.41배의 성능을 가져 온다는것을 보여 준다.

모든 나머지경우들은 하드웨어일관성캐쉬들을 리용하는것을 가정한다.

해제일치성리용은 일관성체계에 비해 35%의 성능개선을 보여 준다. 미리꺼내기추가

는 시간을 44단위로 더 감소시킨다. 가장 좋은 경우는 일관성캐쉬 RC와 MC로 결합하여 리용하는것이다.

맨 오른쪽 시간막대기는 4개 기구들을 모두 결합하여 얻은것이다. 결합결과는 개별적인 캐쉬들을 리용한 경우에 비해 4~7배의 속도개선을 보여 준다.

Stanford에서 보고된 우의 성능평가기준결과들은 일관성캐쉬와 완화형일치성이 모두 성능을 개선한다는것을 보여 준다. 미리꺼내기와 다중문맥으로부터의 성능개선은 눈에 보이는 정도이지만 응용의존성이 훨씬 높다.

여러가지 지연시간은페기구의 결합은 일반적으로 매 기구를 따로따로 적용한것보다 좋은 성능을 얻는다.

다른 말로 4개 기구들은 매개의 효과를 취소하기 보다는 서로 지원한다.

5. 7. 참고문헌주해와 련습문제

작업모임의 개념은 Denning[203]에서 소개되였다.

기억기용량계획작성을 위한 선형계획최량화방책은 Chow[157]에서 서술된다. Cragon[171]은 관흐름처리기설계를 위한 기억기체계를 토론하였다. Przybylski[510]이 쓴 책은 캐쉬기억기구성방식을 취급한다. Flynn[245], Hwang[327], Patterson과 hennessy[305]들의 책들은 순차와 병렬구성방식에서 여러가지 요구에 대한 계층기억기설계를 취급하였다.

MESI snoopy규약은 Pentium처리기에서 쓰인것에 기초한다[31]. Censier와 Feautrier[135]는 캐쉬등록부구조를 제기하였다. Dubois 등은[219] 동기화, 일관성 그리고 다중처리기들에서 사건순서화를 특징 지었다. Smith[568]은 1980년 초까지 개발된 캐쉬기억기들을 평가하였다.

Chaiken 등 [137]과 Agarioal 등 [11]은 MIT Alewife다중처리기에서 등록부에 기초한 규약을 평가하였다.

캐쉬구성방식에서 다른 개발은 Cubois와 Thakkar[220], Stone[590]에서 찾아 볼수 있다.

Adve와 Gharachorloo는 공유기억기일치성모형들의 개괄을 제공하였다[8]. 순차일치성개념은 Lamport에 의해 개발되였다[391]. 약한 일치성기억기모형은 Dubois 등에 의해 제기되였다[218]. TSO와 PSO의 약한 모형들의 형식적고찰은 Sindhu 등에서 주어졌다[558].

Adve와 Hill[7]은 약한 순서화발행을 연구하였다. 처리기일치성모형은 Goodman[273]이 제기하였다.

Stanford해제일치성모형은 Gharachorloo[264] 등에서 찾아 볼수 있다.

상업적완화형기억기모형들은 IBM체계/370 [341], Sun SPARC[579], Cray GiagRing[339], Digital Alpha[563], IBM PowerPC[434]를 위해 연구개발되였다.

4개의 공유기억기모형들이 Chong과 Hwnag에 의해 평가되였다[155]. Stanford Dash다중처리기는 Lenoski 등 [405]에서 보고되였다.

KSR-1은 Burkhardt 등 [116]에서 서술된다.

Swedish DFM은 Hagersten 등 [293]에서 보고된다.

Bisiani와 Ravishanker[83]은 PLUSDSM체계를 개발하였다. Nitzberg와 lo[468]에서 서술되고 TreadMarks는 Rice[30]에서, Wind Tunnel은 Wisconsin[515]에서, Shrimp는

Princeton[237]에서 서술된다.

Stenstrom 등은 CC−NUMA와 COMA구성방식의 성능을 비교하였다. 쏘프트웨어조종미리꺼내기를 통한 지연시간허용은 Mowry와 Gupta[454]에서 연구된다.

Gupta 등은 분산일관성캐쉬들의 효과를 평가하였다[285].

Saavedra 등[522]은 여러가지 자료미리꺼내기방법으로 평가하였다.

다중스레드구성방식의 훌륭한 개괄은 Nikhil[467]에 의해 주어 진다. Saavedra 등[521]은 병렬계산을 위한 다중스레드구성방식을 모형화하였다.

Smith[569]는 다중토막화계산에 관계하는 자료국부성을 고찰하였다. 여러가지 지연시간은폐기술들은 역시 Hwang[327]과 Culler 등 [8]에서 취급된다.

문 제

문제 5.1. 계층기억기설계 및 관리와 관계되는 다음과 같은 기술용어들을 정의하시오.

(1) 가상주소공간
(2) 물리주소공간
(3) 주소이행
(4) 캐쉬행크기
(5) 캐쉬적중률
(6) 페지오유
(7) 기억기교체방책

문제 5.2. 2−준위계층기억기 M_1, M_2을 고찰하시오. M_1의 적중률을 h로 표기하자. C_1과 C_2을 KB당 비용, s_1과 s_2을 기억기용량, t_1과 t_2을 접근시간이라고 하자.

(1) 어떤 조건에서 전체 기억기체계의 평균비용을 값 c_2로 다가가게 할수 있는가.
(2) 이 계층가운데서 유효기억기접근시간 t_a는 무엇인가?
(3) $r=t_2/t_1$을 두개 기억기들의 속도률이라고 하자. $E=t_1/t_a$는 기억기체계의 접근유효성이라고 하자. E를 r와 h용어로 표시하시오.
(4) 그라프종이에서 r=5, 20,100인 경우 h에 대한 E를 곡선으로 표시하시오.
(5) r=100인 경우 효률성이 $E>0.95$이기 위해서 요구되는 적중률 h는 얼마인가?

문제 5.3. 2−준위기억기체계에 대한 용량계획작성문제를 고찰하자. 첫번째 준위에서 M_1은 64KB, 128KB 그리고 256KB의 3개 용량가운데 어느 하나를 가진 캐쉬이다.

두번째 준위에서 M_2은 4KB용량을 가진 주기억기이다. c_1과 c_2은 바이트당 비용, t_1과 t_2은 M_1과 M_2에 대한 접근시간이다. $c_1=20c_2$, $t_2=10t_1$이라고 하자. 3개 용량들에 대한 캐쉬적중률을 각각 0.7, 0.9, 0.98이라고 하자.

(1) 3개 캐쉬설계에서 t_1=20ns일 때 평균접근시간 t_a는 얼마인가?(t_1은 CPU에서 M_1에로, t_2은 CPU에서 M_2에로의 시간이다. M_1에 M_2에로의 시간은 아니다.)

(2) KB당 비용이 c_2=0.2$/KB로 주어 진 전체 계층기억기의 평균비용을 구하시오.

(3) 3개 기억기설계들을 비교하고 평균비용순서, 평균접근시간순서를 각각 밝히시오. 평균비용과 평균접근시간결과에 기초한 최량설계를 선택하시오.

문제 5.4. 다준위계층기억기에서 포함성질과 기억기일관성요구를 설명하시오. 린접한 준위에서 일관성을 유지하는 련속쓰기와 후쓰기를 구별하시오. 또한 계층의 물리기억기와 가상기억기를 유지하는데서 페지와 토막화의 기본개념을 설명하시오.

문제 5.5. 다음과 같은 파라메터값들에 대해 문제 5.3을 다시 고찰하시오. h_1=0.95, t_1=20ns, s_1=512KB, c_1=10$/KB, c_2=5$/KB이다. t_2과 s_2은 미지수이다.

(1) 계층기억기의 전체 비용은 웃한계로서 15000$이다.

(2) 이 기억기체계에서 유효접근시간 t_{eff}를 보여 주는 식을 유도하시오.

(3) 이 기억기체계의 전체 비용을 보여 주는 식을 유도하시오.

(4) 예산제한을 초과함이 없이 M_2이 얼마나 큰 용량(s_2=?)을 얻을수 있는가?

(5) 유효접근시간 t_{eff}=40ns를 얻기 위해서는 주기억기가 얼마나 빨라야 하는가?(t_2=?)

문제 5.6. 국부성에 대하여 다음의 질문에 대답하시오.

계층기억기에서 프로그람/자료접근과 관계되는 시간국부성, 공간국부성과 순차국부성을 설명하시오.

작업모임이란 무엇인가? 작업모임크기에 대한 관찰(감시)창문크기의 민감성에 대하여 해석하시오. 이것은 주기억기적중률에 얼마나 영향을 주는가?

*90−10*법칙과 참조국부성에 대한 관계는 무엇인가?

문제 5.7. p개 처리기를 가진 RISC에 기초한 공유기억기다중처리기를 고찰하자.

매 처리기는 명령캐쉬와 자료캐쉬를 가진다. 매 처리기(두 캐쉬들에서 100% 맞힘률과정)의 최대(peak)실행률은 x MIPS이다. 캐쉬비적중, 공유기억기접근 그리고 동기화부가처리기를 고려하면서 성능식을 유도하려고 한다. 동기화목적으로 평균 @퍼센트의 명령들이 실행되였고 매 동기화동작에 대한 벌칙을 추가적인 $t_2$us이라고 가정하자. 명령당 평균 기억기참조수를 m으로 가정한다.

CPU에 의한 모든 기억기참조들가운데서 f_i 는 명령에 대한 참조비률(퍼센트)이라고 하자. 기계에서 프로그람추적이 긴 주기다음에 명령캐쉬와 자료캐쉬는 각각 적중률 h_i와 h_d를 가진다고 가정하자.

캐쉬비적중에서 명령과 자료는 평균접근시간 t_m을 가진 공유기억기로부터 접근된다.

(1) P, x, m, f_i, h_d, t_m, @, t_s에 의해 다중처리기의 유효MIPS률에 근사한 식을 유도하시오.

(2) m=0.4, f_i=0.5, h_i=0.95, h_d=0.7, @=0.05, x=5, t_m=0.5us, t_s=5us가 주어 졌다. 유효

전체 실행률 25MIPS를 얻는 우의 다중처리기체계에서 필요되는 처리기최소수를 결정하시오. 캐쉬일관성, 접근충돌 혹은 페지오유에 의해 생기는 다른 모든 기억기간섭들은 무시하시오.

문제 5.8. 다음과 같은 비용정보가 주어 졌을 때 문제 5.7을 다시 고찰하자. 모든 캐쉬들과 공유기억기의 전체 비용은 웃한계로서 25000 $을 가진다고 가정한다. 캐쉬기억기비용은 4.70$/KB, 공유기억기비용은 0.4$ /KB이다. p=16처리기들에 대해 매개 처리기는 명령캐쉬 s_i=32KB와 자료캐쉬 s_d=64KB일 때 주어 진 계산제한에서 얻을수 있는 최대공유기억기용량 c_m(MB)은 얼마인가?

문제 5.9. 상태이행도표를 가진 다음의 3개 Snoopy규약을 규정하시오. 이 규약들을 캐쉬상태수가 증가되는데 따르는 3개 캐쉬구성방식에 적용한다. 모든 캐쉬상태, 캐쉬사건 그리고 Snoopy모선사건은 지정되여야 한다.

(1) WT캐쉬들에 대한 두개 상태 SI규약. 여기서 S는 공유상태, I는 캐쉬행의 무효상태이다.

(2) WB캐쉬들에 대한 3개 상태 MSI규약. 여기서 S와 I는 우와 같고 M은 캐쉬행의 변경상태이다.

(3) 결합형WT/WB에 대한 4개의 상태 MESI규약. 여기서 E는 캐쉬행이 단 한번만 씌여 지는 배타적상태이다.

문제 5.10. 공유기억기를 가진 두개 처리기에서 두 프로그람(P_0과 P_1)의 병행실행을 고찰하자. 변수 A, B, C, D는 값 0으로 초기화되고 Print지령문은 두개 변수를 같은 주기에 나누어 인쇄한다고 가정하자.

출구는 ADBC 혹은 BCAD로서 4−taple을 이룬다.

P_0:	P_1:
a. A=1	d. C=1
b. B=1	e. D=1
c. *Print* A,D	f. *Print* B,C

(1) 개별적인 프로그람순서를 보존하는 5개 명령문들의 모든 실행엇끼우기순서를 쓰시오.

(2) 프로그람순서들이 보존되고 모든 기억기접근들이 매우 적다고 가정한다. 즉 한 처리기에 의한 기억은 다른 처리기에 의해 즉시 볼수 있다. 모든 가능한 4조의 출구결합을 쓰시오.

(3) 프로그람순서들이 보존되지만 기억기접근은 적지 않다고 하자. 즉 한 기억기에 의한 기억은 어떤 다른 처리기들이 갱신된 자료를 즉시에 관측(감시)할수 있도록 완충할수 있다. 이 조건에서 모든 가능한 4조의 출구결합을 쓰시오.

문제 5.11. Stanford Dash계층기억기와 등록부규약을 연구한후 성능분석을 통하여 다음과 같은 질문에 대답하시오.

(1) Dash다중처리기구성방식에서 쓰인 캐쉬상태들을 규정하시오.

(2) 계층기억기에서 캐쉬등록부들이 어떻게 실행되는가?

(3) 원격클라스에서 오물인 원격캐쉬불로크를 읽을 때 Dash등록부에 기초한 일관성규약을 설명하시오.

문제 5.12. 확대가능한 다중처리기, 다중콤퓨터 혹은 콤퓨터클라스터를 제작하기 위한 NOMA와 NORMA기억기구성방식을 특징 지으시오.

매 캐쉬에서 전용구성방식특징제작을 설명하는 실례연구로서 실례콤퓨터체계를 선택하시오. 그것들의 장점과 결점을 비교하시오.

(1) CC−NOMA구성방식

(2) COMA구성방식

(3) NCC−NOMA구성방식

(4) 쏘프트웨어실행형DSM구성방식

(5) 성긴결합클라스터 NORMA구성방식

(6) MPP를 위한 밀집결합 NORMA구성방식

문제 5.13. 작거나 큰 작업모임과 낮거나 높은 통신−계산률에서 NUMA와 COMA구성방식의 성능을 비교하시오.

(1) 주어 진 응용프로그람에서 두 구성방식들이 낮은 CCR와 작업모임을 가질 때 왜 잘 수행할수 있는가를 설명하시오.

(2) 큰 작업모임, 낮은 통신−계산률을 가질 때 COMA구성방식이 NOMA구성방식보다 왜 더 잘 수행하는가를 설명하시오.

(3) 작은 작업모임, 높은 통신−계산률을 가질 때 COMA기계는 NOMA구성방식보다 왜 더 나쁘게 수행하는가를 설명하시오.

문제 5.14. 다음의 지연시간허용기술의 기본내용을 요약하여 설명하시오. 쏘프트웨어와 하드웨어방법들을 고찰해야 한다.

(1) 지연시간감소기술

(2) 지연시간회피기술

(3) 지연시간은페기술

(4) 분산일관성캐쉬

(5) 읽기지연시간은페를 위한 자료미리꺼내기

(6) 쓰기지연시간은페를 위한 다중스레드화

문제 5.15. 다중처리기체계의 공유기억기에 대한 다음 4개의 일치성모형에서 기억기

접근제한을 비교하시오.

스레드당 동기화접근은 물론 읽기, 쓰기접근을 취급해야 한다.

(1) Dubois 등에 의해 지정된 약한 일치성(WC)기억기모형
(2) Good man에 의해 지정된 처리기일치성(PC)기억기모형
(3) Lamport에 의해 지정된 순차일치성(SC)기억기모형
(4) Gharachorlco 등에 의해 지정된 해제일치성(RC)기억기모형

문제 5.16. 분산공유기억기인 p개 프로쎄스마디를 가진 확대가능한 다중처리기를 고찰하자.

$1/R$을 호상접속한 망을 통해 원격기억기에 접근요청을 발생하는 매 프로쎄스마디의 비률이라고 하자.

L은 원격기억기접근을 위한 평균지연시간이라고 하자.

다음과 같은 조건에서 처리기효률 E에 대한 식을 유도하시오.

(1) 매 처리기는 단일스레드이고 지연시간은페를 위한 일관성지원 혹은 다른 하드웨어지원이 없이 개별캐쉬만을 리용한다.
(2) 일관성캐쉬가 자료공유를 가진 하드웨어에 의해 지원되고 h는 원격요청이 국부캐쉬에 의해 만족될수 있는 확률이라고 가정하자. E를 R, L, h의 함수로서 표시하시오.
(3) 이제 매 처리기가 N개의 문맥을 동시에 조종할수 있도록 다중토막화되였다고 하자. 문맥−절환부가처리를 C라고 하면 E를 N, R, L, h, C의 함수로 표시하시오.
(4) 이제 $r=p$이고 쌍방향련결인 2차원 $r \times r$원환체(torus)리용을 고찰하자. t_d를 린접한 마디사이의 시간지연, t_m을 국부기억기접근시간으로 하자.

망이 완충이 없이 매 요청에 충분히 빨리 응답한다고 가정하자.

지연시간 L을 P, t_d, t_m의 함수로 표시하시오.

그다음 효률 E를 N, R, h, C, P, t_d, t_m의 함수로 표시하시오.

문제 5.17. 4개 문맥절환방책을 비교하시오.

즉 캐쉬비적중에서 절환, 모든 적재에서 절환, 매 지령주기에서 절환 그리고 명령블로크.

(1) 매 문맥절환방책을 실현하는데 왜 전용하드웨어와 쏘프트웨어지원이 필요한가?
(2) 매 문맥절환방책의 우점과 결점은 무엇인가?
(3) 처리기복잡성과 효률성에 따라 4개 방책들을 정렬하시오.
(4) 어떤 프로그람동작에서 매 방책들을 충분하게 수행할수 있는가?

제6장. 체계호상접속과 기가비트망

이 장에서는 리용가능한 망기술과 체계호상접속에 대하여 고찰한다.

중점은 표준적인 호상접속들과 상품망들 그리고 그 설계와 응용원리들에 두고 있다.

그리고 위상들과 규약들, 모선들의 접속, 크로스바교환기들, 다단교환기들, 매체공유망들, 세포절환 및 파케트교환망들을 고찰한다.

이 내용들은 다음장들에서 병렬 또는 클라스터콤퓨터의 구조를 고찰하는데 필요하다.

6. 1. 호상접속망의 기초

교환구조들은 현대 콤퓨터망의 핵심부이다. 아래에서는 기본적인 망술어들과 환경들, 분류, 표준규약들, 현재의 고대역너비망들에 대한 성능척도들을 소개한다.

이 책에서 망은 여러 콤퓨터들사이의 접속으로 리해한다.

체계호상접속은 통합콤퓨터체계의 부분체계들을 하나로 결합한다. 다중콤퓨터클라스터는 두가지 형태의 호상접속구조를 사용하여야 한다.

6. 1. 1. 호상접속환경

확대가능한 다중처리기나 다중콤퓨터클라스터, 분산체계에서 그 구성요소(탁상호스트와 봉사기, 교환기, 망, 적응기기판과 주변장치 등)들은 5개의 망환경들에 호상접속된다.

체계 또는 I/O모선이나 크로스바/다단교환기들은 콤퓨터가동환경의 단일시렁이나 같은 방에서 호상접속된 몇개의 시렁들내로 한정된 체계호상접속들이다.

여러 망기술들을 그림 6-1에서와 같이 2차원공간에 배치할수 있다.

체계망(System-Area Network, SAN)은 처리기들과 기억기, 대면부기판들을 짧은 거리내에서 마이크로스트리프(microstrip)케블로 접속한다.

국부망(LAN)은 건물이나 구내 또는 기업소범위로 제한된다.

지역망(MAN)은 전체 지역이나 한개 도시를 포함한다.

광지역망(WAN)은 흔히 여러개의 보다 작은 망들의 호상접속으로 보는데 따라서 긴 거리로 확장할수 있다.

이 망기술들은 다음부분들에서 서술된다.그림 6-1은 1997년의 기술에 기초하고 있다.

최대망거리는 왼쪽에서 오른쪽으로 수평축을 따라 증가한다.

호상접속환경들은 동작거리에서 일부가 겹칠수 있다. 겹치는 현상을 고려하여 그림 6-1에서 거리규모는 표식하지 않았다.

모선들과 망에 대한 술어들의 모임을 정의하는데 그림 6-2가 리용된다. 이것은 공업과 여러 대학에서 널리 리용된다. 처리기소편의 판들은 처리기모선을 형성한다.

처리기와 기억모듈들을 련결하는 모선은 **국부모선** 또는 **기억모선**이라고 부르는데 그

100 Gbps
10 Gbps
1 Gbps
100 Mbps
10 Mbps
망 대 역 너 비
MIN 또는 크로스바 교환기들
국부 모선들
SCI
HiPPI
Myrinet
기가비트 Ethernet
I/O 모선들
빛섬유통로
FDDI
ATM
고속이써네트: 100 Base T
IsoEnet
이써네트: 10 Base T
Buses or Switches
SANs
LANs
MANs
WANs

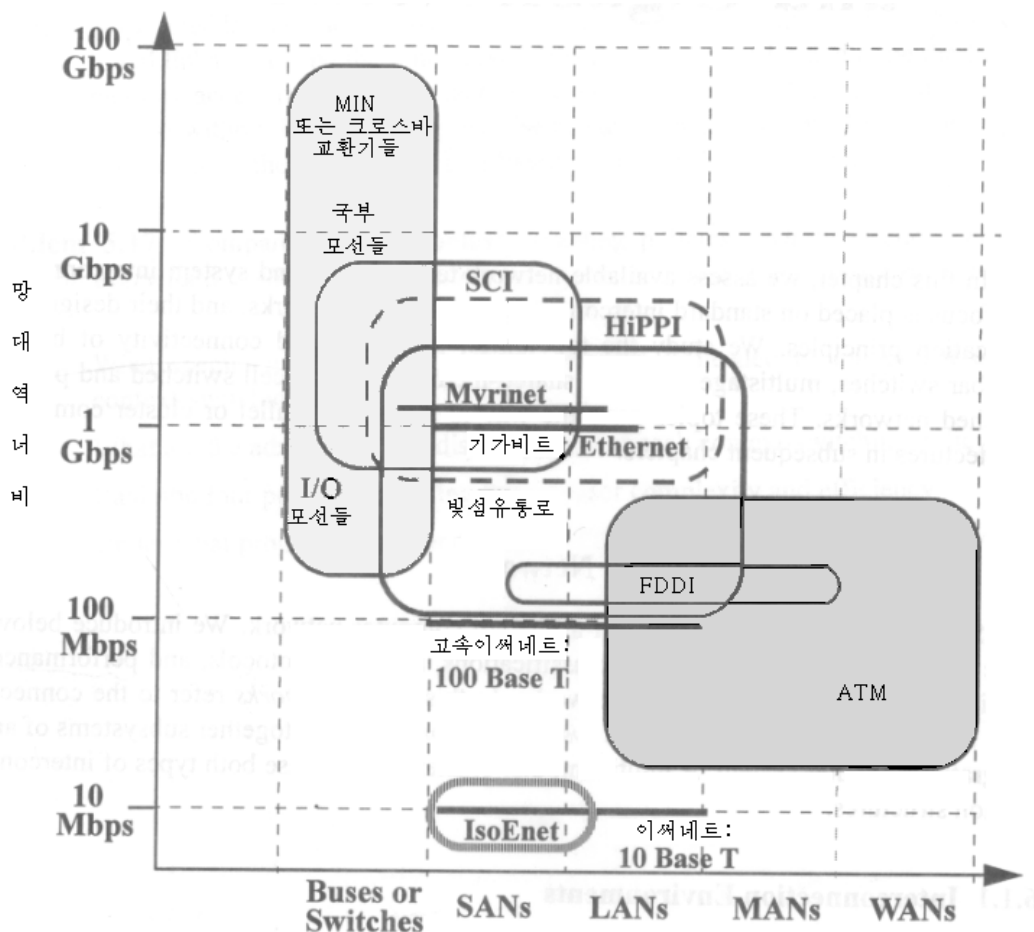



그림 6-1. 확대가능한 병렬 및 클라스터가동환경들을 구성하기 위한 체계호상접속과 망기술들

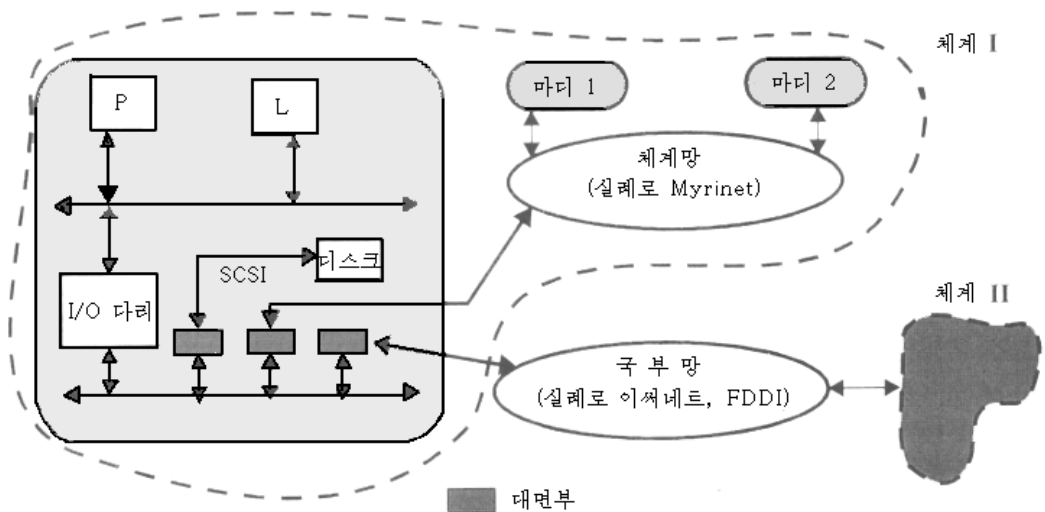



그림 6-2. 국부모선(기억기모선)과 체계모선(I/O 모선), 체계국부망에 대한 설명

길이는 10cm내에 있다.

체계모선(또는 I/O모선이라고 부른다.)은 디스크구동기, 테프구동기, 망대면부기판과 같은 I/O장치들을 련결하기 위한 확장홈들을 제공한다.

확장홈은 국부모선과 I/O다리회로를 통하여 접속된다. 체계모선은 흔히 모판이나 체계시렁의 뒤쪽 판 또는 중심판에 내장되며 따라서 그 한계는 2m이다.

체계모선의 실례로서 AT모선(ISA), EISA모선, PCI모선 그리고 IBM PC들에서 사용되는 Micro Channel, Sun 워크스테이션들의 Sbus를 들수 있다.

술어 I/O모선은 종종 SCSI(Small Computer System Interface)모선을 가리킨다는데 주의하여야 한다.

케블기술에 따라 내장 I/O모선은 3m를 넘지 않으며 외부 SCSI케블은 약 20m까지 확장될수 있다.

많은 다중처리기들과 다중콤퓨터체계들은 자기의 하드웨어를 몇개의 마디들로 나눈다.

단일체계를 형성하기 위하여 이러한 마디들을 하나로 접속하는 망을 **체계망**(SAN)이라고 부른다. 대조적으로 국부망은 여러 체계들을 접속하는데 사용된다. 물론 기술(실례로 이써네트)은 체계망들과 국부망들에서 사용될수 있다. 체계의 구성은 9장에서 명백히 고찰하므로 여기서는 클라스터들과 단일체계영상을 상세히 서술한다.

체계망은 3~25m범위에 있다.

국부망은 25~500m범위에 있으며 2km까지 확장할수 있다.

지역망은 보통 25km까지 또는 그이상으로 확장할수 있다.

광대역망은 본질에 있어서 거리에 한계가 없다. 이것은 국제적인 령역으로 확장할수 있다.

수직축은 단위시간에 호상접속 또는 망을 통하여 전송가능한 정보의 최대량으로 정의되는 망대역너비의 증가에 대응한다.

일부 전문가들은 이것을 간단히 **망의 속도**라고 부른다. 흔히 Mbps(Million bits per seconds) 또는 Gbps (Giga bits per seconds)로 측정한다.

그림 6-1에서 보여 준 호상접속기술들은 대역너비가 10Mbps로부터 100Gbps까지의 범위에 있다.

다음부분들에서는 모선들과 크로스바교환기 그리고 여러가지 망기술들의 속도를 고찰한다.

앞으로 특정한 기술에 의해 대역너비가 높아 지고 케블거리는 증가될것이다. 그리하여 그림 6-1에서 둘러 싸인 령역들중의 일부는 웃쪽의 오른쪽 구석을 향하여 이동할것이다.

Tbps(Tera bits per seconds)망속도는 2000년에는 훨씬 높아 질것이다.

6. 1. 2. 망구성요소

모든 교환망들은 기본적으로 세개의 구성요소 즉 련결(통로 또는 케블), 교환기(경로기), 망대면부기판들로 구성된다.

매체공유망들은 교환기를 사용하지 않는다. 망의 다른 특징들을 보기전에 아래에서 망구성요소들을 잠간 소개한다.

련결 또는 통로, 케블

3개의 술어들은 모두 콤퓨터체계안에서 두개의 하드웨어단위들사이의 물리적련결을 의미한다.

련결은 동선이나 빛섬유케블로 실현할수 있다. 가장 단순한 련결은 동선으로 된 비차폐꼬임쌍선(UTP)이다. 동선련결은 값이 눅지만 신호법문제로 해서 짧은 케블길이로 제한된다. 케블길이를 차폐꼬임쌍선(STP)을 사용하면 일정한 정도로 확장할수 있다. 빛섬유케블은 값은 비싸지만 매우 높은 대역너비와 보다 긴 케블길이를 제공한다.

일부 저자들은 련결을 통로나 **케블**이라고 부른다.

련결은 두개의 교환기를 련결하거나 교환기와 호스트마디에 설치된 망대면부를 련결할수 있다. 짧은 련결은 한번에 오직 하나의 론리신호만을 포함하며 한편 긴 련결은 전송선처럼 론리신호들의 렬을 같은 시각에 선로를 따라 넘기기시킨다.

직렬련결이나 얇은 련결은 다중화된 방식에서 자료와 조종신호들에 의해 공유되는 1bit선로를 가진다.

넓은 련결이나 병렬련결은 여러 bit선로들을 가지며 자료와 조종정보의 병렬전송을 가능하게 한다. 련결은 두가지 박자와 기구들중의 하나 즉 동기 또는 비동기에 의해 조종된다.

동기박자화는 원천과 목적말단(End)이 같은 대역적인 박자로 동작한다는것을 의미한다.

비동기련결은 두끌이 서로 다른 박자렬로 련결(손 잡도록)되도록 하는 어떤 매몰된 박자부호화기구를 사용한다.

교환기

교환기는 교환망을 구성하는데 필요하다.

일반적으로 교환기는 여러개의 입력 및 출력포구를 가진다. 매 입력포구는 도착한 파케트나 세포들을 처리하는 하나의 수신기와 입력완충기를 가진다. 매 출력포구는 나가는 자료신호를 다른 교환기나 망대면부에 접속된 통신선로로 보내는 하나의 송신기를 가진다. 4개의 입력, 4개의 출력교환기를 그림 6-3에 보여 준다.

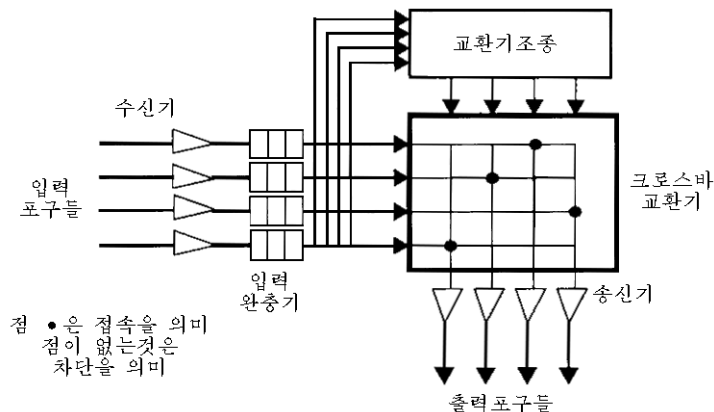



그림 6-3. 4개의 입력와 4개의 출력사이의 임의의 치환접속을 제공하는 4개 포구 크로스바교환기

내부크로스바는 n개 입력, n개 출력사이의 n개 접속을 동시에 확립하는데 리용된다. 매개 교차점을 프로그람의 조종하에서 접속(on) 또는 비접속(off)으로(그림 6-3에서 점으로 표식한것과 같이) 절환될수 있다.

이 수자 n을 흔히 **교환기의 차수**라고 한다.

여러개의 교환기들과 련결들이 선택된 위상에 따라 큰 교환망을 구성하는데 리용된다.

망대면부회로(Network Interface Circuitry)

망대면부회로(NIC)는 호스트콤퓨터를 국부망이나 다른 망에 접속하는데 리용된다. 그래서 일부 전문가들은 이것을 **호스트대면부**라고 부른다. NIC는 호스트와 망사이의 쌍방향통신흐름을 처리할수 있어야 한다. 따라서 NIC의 구조는 망과 호스트에 의존한다. 서로 다른 호스트들은 같은 망에 접속되여 있을 때조차도 서로 다른 대면부기판을 요구할수 있다.

전형적인 NIC는 내장된 처리기와 약간의 입력 및 출력완충기, 약간의 조종기억과 론리를 포함한다. NIC는 파케트 또는 세포의 형식화와 경로선택, 간섭성검사, 흐름 및 오유의 조종 등의 기능을 수행한다.

그러므로 NIC의 가격은 포구의 크기와 기억용량, 처리능력, 조종회로에 따라 결정되며 종종 경로교환기보다 더 큰 복잡성을 일으킨다.

6. 1. 3. 망의 특징

여기서는 기본적인 망술어들을 구별한다.

흔히 이 술어들은 두개의 반대되는 특징들을 가진 쌍들로 나타날수 있다.

또는 매개 특징을 전체 망설계공간의 차원으로 간주할수 있다.

정적망 대 동적망

접속과 조종방안에 기초하여 호상접속망을 여러가지 종류로 분류할수 있다. 정적망은 프로그람이 실행되는 동안 변화시킬수 없는 점대점(point to point)련결들로 구성된다.

다시 말해서 정적망들은 프로쎄스마디들사이의 고정된 접속들을 가진다. 정적망은 하나의 호스트콤퓨터만이 매개 마디의 교환기에 접속되므로 직접망이라고도 불리운다.

한편 동적망은 교환통로들에 의해 실현되는데 사용자프로그람의 통신요구에 맞게 동적으로 구성된다.

동적망은 체계모선과 크로스바교환기, 다단망들을 포함하며 여러가지 체계망과 국부망, 지역망, 광대역망을 포함한다. 동적망은 간접망으로도 불리우는데 여러 호스트들이 특정한 마디의 교환기에 접속되며 경로정하기는 교환결정들의 렬에 의해 진행된다. 두가지 형태의 마디들을 다 가지는 일부 혼합망은 프로그람의 조종하에서 정적접속뿐아니라 동적접속도 지원한다.

매체공유망

매체공유망에서 물리적련결(동 또는 섬유)들은 망에 접속된 모든 콤퓨터들에 의하여

호출될수 있다. 이러한 리유로 매체공유망은 다중접근망으로도 알려 져 있다. 그러나 한번에 오직 하나의 요청자만이 허용된다.

다시 말해서 대역너비는 여러 요청자들에게 공유된다. 매체공유망설계에서 가장 곤난한 문제는 경쟁하는 모든 요청자들에게 같은 대역너비를 배정하는것이다.

매체공유망은 FDDI(빛섬유자료대면부)와 고속이써네트(100base-T), 빛섬유통로기술로 실현할수 있다. 따라서 교환기는 사용되지 않으며 매체공유망은 흔히 대응하는 교환망보다 값이 눅다.

모든 요청자들에게 리용가능한 대역너비를 배정하는것을 대역너비관리문제라고 한다.

망접근규약의 선택은 대역너비관리체계의 효률에 큰 영향을 준다.

교환망

대조적으로 교환망은 한번에 한명의 요청자에게 매체자원을 배정 또는 해제하기 위하여 교환구조(switching fabric)를 사용한다. 다시 말해서 매체련결에로의 접근은 한번에 한명이상의 사용자에게 공유되지 않는다.

많은 매체공유망들은 비용을 증가시켜 교환설계로 전환할수 있다.

실례로 대부분의 이써네트와 FDDI고리들은 매체를 공유하지만 교환식이써네트 또는 교환식FDDI고리들도 가능하다.

한편 모든 비동기전송방식(ATM)망들은 고유한 교환망이다. 교환망들은 흔히 회선교환, 파케트교환, 세포교환으로 설계된다.

회선교환망

교환은 점대점통신을 하는 마디쌍들사이 또는 집체통신을 하는 여러 마디들사이의 경로를 정하기 위하여 필요하다. 회선교환에서 원천마디로부터 수신마디까지의 전체 (련결들과 완충기들의)경로는 전체 전송기간에 예약된다.

이것은 예약된 경로에서의 긴 렬차에 의한 철길수송과 류사하다. 여기서 그 어떤 다른 렬차도 동시에 같은 경로를 공유하지 못한다.

만일 기차가 대단히 길면 다음기차가 통과할수 있게 되기까지 긴 시간이 걸릴것이다.

파케트교환망

파케트망에서 긴 통보문은 작은 파케트들의 렬로 쪼개진다.

매개 파케트는 경로정하기정보와 유효자료토막을 포함한다. 같은 통보문의 파케트들은 각각 서로 다른 경로로 로정을 정할수 있다. 이것은 렬차 한대분의 전체 상품을 나누어 작은 화물차들로 나르는것과 비슷하다. 파케트교환은 전송전에 통보문을 파케트들로 쪼개고 모든 파케트들이 도착한후 통보문을 다시 조립할것을 요구한다.

파케트교환은 망자원을 더 잘 리용할수 있게 한다. 때문에 서로 다른 통보문의 많은 파케트들이 같은 망련결들과 완충기들을 더 효과적으로 공유할수 있다.

세포교환망

파케트망에서 서로 다른 통보문의 파케트들은 길이가 다를수 있다.

다시 말해서 긴 파케트들의 렬에 길고 짧은 파케트들이 련재될수 있다. 긴 파케트에 뒤따르는 작은 파케트는 공유된 경로에서 긴 파케트가 제거될 때까지 긴 지연을 받게 된다.

세포교환은 긴 파케트를 고정된 크기의 작은 세포들로 분할함으로써 파케트교환을 개선하였다.

분할의 목적은 그림 6-4에 묘사된바와 같이 긴 파케트들에 의해 부가된 차단을 완화하는것이다.

실천에서 긴 파케트는 4096B정도이며 짧은 파케트는 단지 몇바이트이다. 세포교환망에서 대표적인 세포는 56B이다. 세포망에서는 고정된 전송지연이 가능하다.

파케트교환망에 비한 세포교환망의 또 다른 우점은 가변길이의 파케트들대신에 작고 고정된 크기의 세포들을 다루는것으로 하여 세포교환기의 장치설계가 간단해 진다는데 있다.

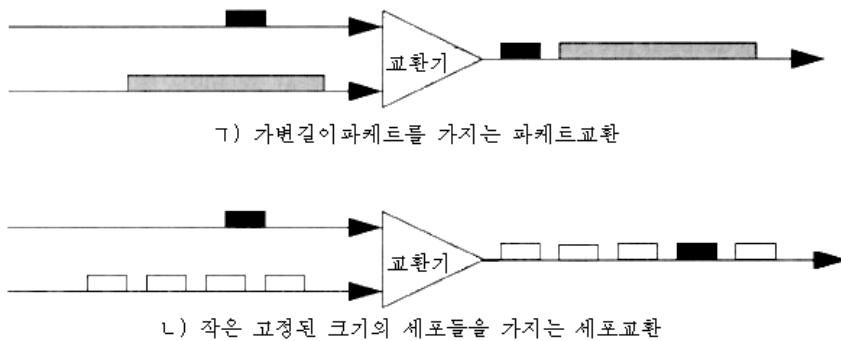



그림 6-4. 두가지 형태의 교환망들에서 파케트와 세포들의 련재

세포교환은 ATM망의 도입으로 대중화되였다.

자료통보문을 많은 세포로 쪼개는것은 대단히 낮은 망성능을 초래할수 있다. 왜냐하면 전송도중에 세포가 잃어 질수 있고 재전송은 흔히 허용되지 않기때문이다.

이러한 문제들은 ATM세포교환기와 ATM망이 론의되는 6.5절에서 취급한다.

6. 1. 4. 망성능척도

지연과 대역너비는 망이나 체계호상접속의 성능을 평가하는데 사용되는 두개의 기본적인 척도이다.

아래에 이 두가지 성능척도의 여러가지 측면을 서술한다.

통신지연(Communication Latency)

원천마디로부터 목적마디까지 통보문을 보내는데 요구되는 전체 시간이다.

이 지연은 4개의 요소시간들 즉 망의 량끝에서 통보문을 보내고 받는것과 관련된 쏘프트웨어부가처리, 통로점유에 의한 통로지연(또는 전체 통보문길이를 통보대역너비로 나눈것), 정해 진 경로를 따라 경로선택들의 렬을 만드는 련이은 교환기들에서 소비된 시간

에 의한 경로정하기, 지연, 망에서 통신흐름경쟁에 의한 경쟁지연으로 이루어 진다.

쏘프트웨어부가처리는 호스트핵심부가 량끝에서 통보문을 처리할 때 크게 기여한다. 통로지연은 흔히 병목련결 또는 병목통로에 의해 결정된다.

경로정하기지연은 경로정하기거리(또는 경로길이나 량끝사이의 hop의 수)에 비례한다.

경쟁지연은 망에서의 통신흐름조건에 의존하므로 예견하기가 가장 힘들다.

망지연(Network Latency)

쏘프트웨어부가처리와 경쟁지연은 다 프로그람의 동작에 의존한다. 그러므로 하드웨어설계자들은 두개의 술어 즉 통로지연과 교환지연의 합을 망지연으로 간주한다.

이 량은 프로그람의 동작과 통신흐름조건과는 독립인 망하드웨어특징에 의해 완전히 결정된다.

가벼운 통신흐름(경쟁이 없는)통보문넘기기망체계에서 망지연(보통 1㎲근방)은 쏘프트웨어부가처리나 경쟁지연보다(1㎲의 10 또는 100배로) 훨씬 작다.

전체적으로 통신지연은 량끝에서 핵심부부가처리와 (머리부를 포함하는) 통보문길이, 통로대역너비, 교환지연(또는 사용된 경로정하기알고리듬), 경로길이, 망통신흐름(또는 프로그람동작)의 함수이다.

지연을 줄이기 위한 많은 노력이 쏘프트웨어부가처리시간을 줄이거나 은페시키는데로 지향되고 있다.

여러가지 지연감소나 지연은페기술들은 이 장의 마지막에서 론의된다.

포구당 대역너비(Per−Port Bandwidth)

앞에서 이미 포구당 대역너비를 망의 임의의 포구에서 임의의 다른 포구로 단위시간에 전송할수 있는 최대비트(또는 바이트)수로 정의하였다.

실례로 IBM SP2고성능교환기(HPS)는 포구당 40MB/s의 대역너비를 가진다.

균형망에서 모든 포구는 위상적으로 동등하다.

다시 말하여 포구당 대역너비는 균형망안의 포구위치와 독립이다. 그렇지 않은것은 비균형망이다.

비균형망에서 포구당 대역너비는 모든 포구당 대역너비들의 최소값으로 정의한다.

총 대역너비(Aggregate Bandwidth)

주어 진 망의 총 대역너비는 마디들의 한쪽 절반으로부터 다른쪽 절반으로 초당 전송할수 있는 최대 bit(또는 byte)수로 정의한다.

실례로 HPS는 n개마디(포구)들로 이루어 진 균형망인데 n은 512가 웃한계이다.

포구당 대역너비가 40MB/s일 때 512개 마디를 가진 HPS의 총 대역너비는 (40×512)/2=10.24GB/s로 계산된다.

2로 나누기는 쌍방향통신흐름을 한번만 계산되여야 한다는 사실에 기초하고 있다.

2등분대역너비(Bisection Bandwidth)

집합대역너비는 2등분대역너비라는 위상적인 술어에도 관계된다.

2등분대역너비는 망을 똑같이 둘로 나누는 2등분우의 모든 선들을 지나가는 초당 최대bit(byte)수로 정의한다.

*b*는 2등분을 지나가는 련결들의 수, *w*는 련결당 선의 수(련결폭 또는 **통로폭**이라고 부른다.)라고 하자.

적 b×w를 2등분폭이라고 부르는데 이것은 2등분면을 지나가는 전체 선의 수를 나타낸다.

매 선이 *r* b/s로 전송한다면 2등분대역너비는 B=*b*×*w*×*r* b/s로 정의된다. 망을 거쳐 M byte를 움직이는 시간은 M/B이하이다.

일반적으로 망대역너비는 망위상과 망크기(또는 포구수), 통로수, 교환기의 차수, 망박자속도에 민감하다. 하드웨어구조는 망대역너비에만 영향을 주며 프로그람의 동작, 통신흐름형태와 독립이다.

이 독립성은 망지연에 대해서도 성립한다.

기계와 프로그람동작은 통신지연에 영향을 준다.

6. 2. 망의 위상구조와 속성

기본적인 망속성들은 구조적인 련관속에서 밝혀 진다.

망위상을 조사하기전에 망의 복잡성, 통신효률, 가격을 평가하는데 자주 리용되는 파라메터들을 정의한다.

6. 2. 1. 위상구조속성과 기능속성

호상접속망은 기본적으로 가지로 련결된 유한마디들의 방향 또는 비방향그라프로 표현된다. 마디는 하나의 교환기나 호스트장치일수 있다.

가지는 련결이나 통로로 볼수 있다.

그라프마디의 수는 망의 크기에 대응된다.

마디차수(Node Degree)

마디로 들어 오는 가지(련결이나 통로)의 수를 마디차수 d라고 한다.

단일방향통로의 경우 마디안으로 들어 오는 통로의 수는 입력차수이며 마디의 밖으로 나가는 통로의 수는 출력차수이다.

마디차수는 이 두 차수의 합이다.

이것은 매 마디에 요구되는 I/O포구들의 수와 같으며 따라서 마디의 가격으로 된다. 마디차수는 포함되는 가격을 줄이기 위하여 가능한껏 작게 유지하여야 한다.

망직경(Network Diameter)

망의 직경 D는 임의의 두 마디사이의 최대경로길이이다.

경로길이는 통과한 련결의 수로 잰다.

망직경은 임의의 두 마디사이의 명백한 hop의 최대수를 가리키며 따라서 자료망에서의 통신지연을 평가하는 간단한 지표로 된다.

차단망(Blocking Networks) 대 비차단망(Nonblocking Networks)

망에서 통보문넘기기는 차단 또는 비차단으로 설계될수 있다.

차단망은 통보문이 여러 통보문과 충돌하는 경우 후에 전송하기 위하여 완충시킨다. 그러므로 차단망은 완충망과 호상 바꾸어 쓸수 있다.

비차단망은 완충기를 사용하지 않는다. 가변적인 경로들이 통보문충돌을 해소하는데 사용된다. 대부분의 차단망들은 통보문의 경로를 정하기 위하여 교환기를 리용한다.

동기망(Synchronous Networks) 대 비동기망 (Asynchronous Networks)

동기망에서 송신기와 수신기는 시공간적으로 동기화되여야 한다.

이것은 호출자와 응답자사이의 동기화를 요구하는 전화망과 류사하다. 대부분의 동기망들은 완충기를 리용하며 따라서 차단망이다. 비동기망은 완충기를 사용하지 않으며 본질에 있어서 비차단망이다.

비동기망에서는 송신기와 수신기의 동기가 필요 없다. 이것은 우편배달망과 비슷하다. 일부 통신망들은 동기 및 비동기통신을 다 지원하도록 설계되고 있다.

기능성과 복잡성(Functionality and Complexity)

기능성은 파케트경로정하기, 새치기처리, 고속동기, 통보문결합, 자료일관성, 교환기충돌, 통로충돌, 흐름조종, 고장허용, 잃어 버린 파케트처리, 자료미리꺼내기, 지연허용, 대역너비관리 등을 지원하기 위하여 망에 설치된 특수한 특징들 또는 기구들이다.

복잡성은 망구성비용에 영향을 준다. 복잡성은 흔히 망구성에 요구되는 포구, 케블, 교환기, 완충기, 접속기, 중계기들과 대면부론리에 의하여 결정된다.

조종회로, 완충기억기, 전원공급도 망복잡성의 부분이다.

균형성과 확대가능성

망의 임의의 마디로부터 위상이 같게 보이면 그 망을 **균형**이라고 하고 그렇지 않으면 **비균형**이라고 부른다.

균형망들은 VLSI(초대규모집적회로)회로나 PC기판우에 배치하기가 쉽다. 균형통신망에서는 또한 통보문경로정하기도 쉽다.

망의 확대가능성은 망의 모듈적인 확대가능성이다.

확대는 예견된 성능에 비례되게 증가시켜야 한다. 망의 확대가능성은 마디차수와 직경, 균형성, 2등분대역너비, 복잡성과 같은 위상적인 속성들뿐아니라 묶기와 전원분산, 랭각, 프로그람식조종, 통신흐름관리 등과 같은 물리적인 위상 및 관리요소들에 의해 제약된다.

6. 2. 2. 경로정하기방법과 기능

여기서는 교환망의 여러가지 파케트경로정하기방법들을 소개한다.

축적전진방법은 1세대통보문넘기기다중콤퓨터에서 사용되였다.

Cut-through 또는 wormhole경로정하기방법은 오늘 더 많은 다중콤퓨터 또는 클라스터체계들에서 실현되고 있다.

축적전진경로정하기(Store−and−Forward Routing)

전체 파케트는 나가는 련결로 전진하기전에 마디안의 파케트완충기에 기억되여야 한다. 그러므로 련속적인 파케트들은 시간상에서 겹치지 않고 련속적으로 전송된다.

그림 6−5 ㄱ)에서는 마디 1로부터 두개의 중간마디(마디 2와 마디 3)를 거쳐 4개의 세포로 된 파케트를 마디 4로 전송하는 과정을 보여 준다.

매개 파케트는 4개의 세포로 구성된다.

머리부는 *h*로 표시하고 3개의 자료세포는 *a*, *b*, *c*로 표시한다.

머리부는 경로를 정하는데 리용된다. 매개 마디는 파케트를 다음마디로 보내는데 4개 시간단위(파케트의 길이와 같다.)를 소비한다.

따라서 정해 진 경로에서 자원충돌이나 교착현상이 없으면 N1로부터 N4로 파케트를 보내는데 16개의 시간단위가 걸린다. 망이 혼잡되면 경로정하기시간이 더 길수도 있다.

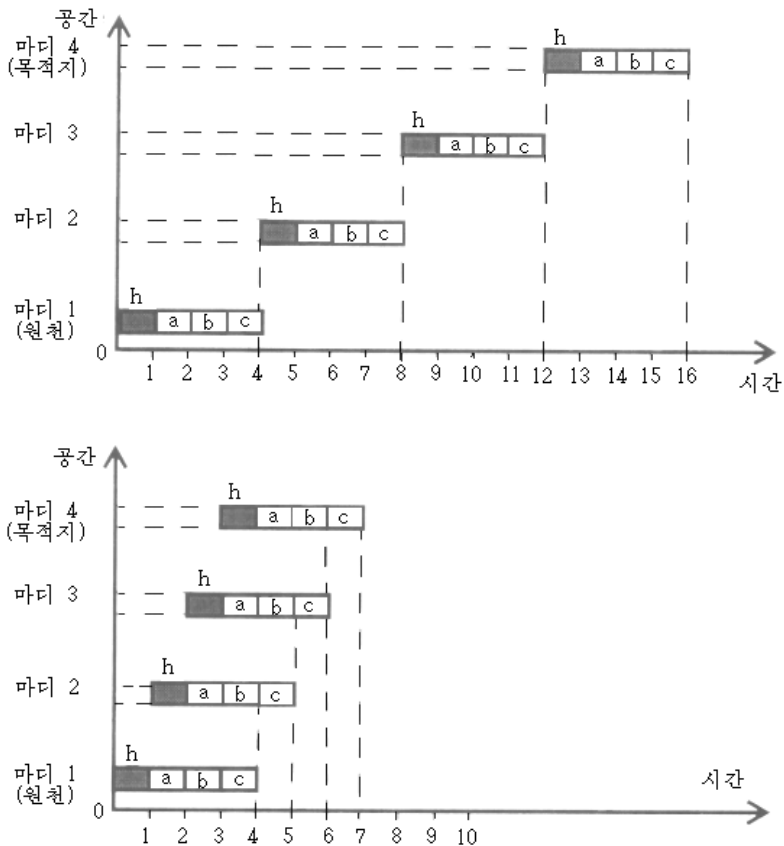



그림 6−5. 파케트교환망에서 축적전진경로정하기와 cut−through 경로정하기
(h : 머리부, *a* ,*b*, *c*:자료요소)

Cut−Through경로정하기

매개 마디는 한번에 하나의 flit, 하나의 세포를 가지는 flit완충기를 사용한다.

일단 머리부가 해득되면 flit는 나가는 련결로 자동적으로 전진한다. 같은 파케트의 모든 자료 flit들은 머리부가 통과하는 같은 경로를 통과한다. 그러므로 매개 마디는 파케트를 단위시간당 한 flit씩 경로로 전진시킨다.

련속적인 파케트들은 관흐름형태로 전진한다.

그림 6-5 ㄴ)의 마디 4가 4개의 flit로 된 전체 파케트를 수신하는데 7개의 시간단위밖에 걸리지 않지만 축적전진경로정하기에서는 필요한 16개의 시간단위가 필요하다.

L을 파케트길이, N을 원천지에서 목적지까지의 마디들의 수라고 할 때 일반적으로 축적전진방법에서는 N×L시간단위가 걸리며 한편 wormhole경로정하기에서는 N−1개의 hop들로 된 경로를 통과하는데 L+N−1개 시간단위만 걸린다.

속도증가는 NL/L+N−1로 기대된다.

충분히 긴 파케트에 대하여 wormhole경로정하기의 속도증가한계는 일반적인 경로정하기방법보다 N배 더 빠르다.

wormhole경로기는 선택된 경로의 망자원(련결, 교환기. 망대면부)들을 사용하여 전송관흐름과 같이 련속적인 파케트들의 전송을 중첩시킨다(그림 6-5).

파케트는 복사와 기억지연이 없는 중간마디를 무시한다.

wormhole경로정하기는 접속지연을 축적전진방법의 500㎲로부터 상업교환기설계의 20㎲로 줄인다.

자료경로정하기기능

자료망은 마디호상간의 자료통신에 리용된다.

아래에 경로정하기망에서 실현되고 있는 요소적인 자료경로정하기기능들을 서술한다.

자료경로정하기기능에는 밀기, 회전, 치환(1대 1), 방송(1대 전체), 집단내방송(n대 n), 개인통신(1대 n), 혼합, 교환(exchange) 등이 있다.

치환(Permutations)

n개의 대상들에 대하여 $n!$개의 치환이 가능하다. 모든 치환들의 모임은 구성(composition)연산의 견지에서 치환군을 형성한다.

치환을 표시하는데 군론의 묶음표기법을 사용할수 있다.

실례로 치환 $\pi = (a, b, c)(d, e)$는 쌍대넘기기 즉 $a \to b, b \to c,\ c \to a,$ $d \to e$를 의미한다.

묶음 (a, b, c)는 주기(Period) 3을 가지며 묶음 (d, e)는 주기 2를 가진다.

이 두 주기를 결합하면 치환은 모두 2×3=6개의 주기를 가진다.

만일 6개의 항목에 치환을 적용하면 항등넘기기 $I = (a),(b),(c),(d),(e)$가 얻어 진다.

임의의 치환형태를 실현하는데 크로스바교환기를 리용할수 있다. 다단망은 망을 통하여 하나 또는 여러개의 통로들에서 일부 치환들을 실현할수 있다.

치환들은 또한 밀기 또는 방송연산들에 의해 실현될수 있다.

n이 클 때 치환속도는 자료경로정하기망의 기본적인 성능을 결정하게 된다.

완전혼합(Perfect shuffle)

완전혼합은 병렬처리를 응용하기 위하여 Harold Stone이 제기한 특수한 치환기능이다. 완전혼합넘기기를 그림 6-6에 보여 준다.

ㄱ) 완전혼합 ㄴ) 반전된 완전혼합

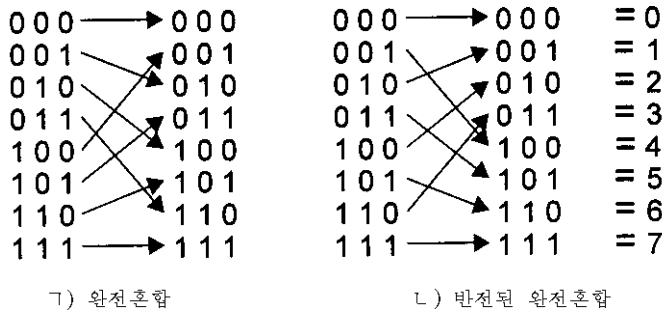
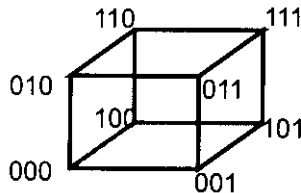



그림 6-6. 8개의 대상에 대한 완전혼합과 거꿀넘기기

n개의 카드(대상)의 혼합은 추기에 의해 고르롭게 진행된다. k비트의 2진수 $X=(X_{k-1},\cdots,X_1,X_0)$로 $n=2^k$개의 카드를 가진 트럼프의 매 카드를 표시할수 있다. 완전혼합은 카드 x를 y로 넘긴다.

여기서 $y=(X_{k-2},\cdots,X_1,X_0,X_{k-1})$는 x를 왼쪽으로 1bit 밀고 가장 낮은 자리로 겹쳐 넣음으로써 얻어 진다.

이것은 완전혼합을 생성하는 2진방법이다.

ㄱ) 2진 $C_2C_1C_0$에서 작성된 8개마디를 가지는 2진립방체

ㄴ) 변환비트 C_0에 의한 경로정하기

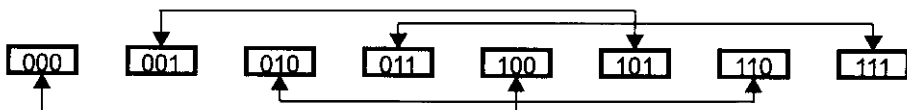
ㄷ) 변환비트 C_1에 의한 경로정하기

ㄹ) 변환비트 C_2에 의한 경로정하기

그림 6-7. 2진립방체에서 3개의 경로정하기기능

하이퍼립방체경로정하기기능(Hypercube Routing Funcion)

그림 6-7 ㄱ)에 3차원 2진립방체를 보여 준다.

3개의 경로정하기기능은 마디주소의 3개 bit에 의해 정의된다.

실례로 그림 6-7 ㄴ)와 같이 가장 낮은 bit인 C_0이 서로 다른 린접한 마디들사이에 자료를 교환할수 있다.

이와 비슷하게 다른 두개의 경로정하기형태는 중간 bit인 C_1 (그림 6-7 ㄷ)과 가장 높은 bit인 C(그림 6-7 ㄹ)를 각각 검사함으로써 얻을수 있다.

방송과 집단내 방송(Broadcast and Multicast)

방송은 1대 전체 넘기기이다.

이것은 방송모선을 사용하는 SIMD콤퓨터에서 쉽게 실현할수 있다.

통보문넘기기다중콤퓨터는 방송통보문들에 집단내 방송기구를 사용한다.

집단내 방송은 마디들의 한 부분모임으로부터 다른 부분모임에로의 넘기기(n대 n)에 대응한다.

개인방송은 선택된 수신자들에게만 개인통보문을 보낸다.

방송은 다중콤퓨터에서 흔히 대역연산으로 취급된다.

6．2．3．망위상구조

정적망들은 일단 접속되면 고정되는 점대점련결들을 사용한다. 이 형태의 망은 흔히 매체를 공유하며 예측가능한 통신흐름형태들에 더 적합하다.

이러한 위상들은 또한 교환망을 구성하는데도 적용될수 있다.

선형배렬(Linear Array)

이것은 N개 마디들이 N개 련결로 접속되여 단일한 폭포모양(cascade)을 형성하는 가장 간단한 망이다(그림 6-8 ㄱ).

중간마디들은 차수가 2이다.

직경은 $N-1$이다. 2등분 폭 b는 1이다.

구조는 균형이 아니며 N이 매우 클 때 통신효률이 문제로 된다. $N=2$인 작은 경우는 선형배렬을 실현하는것이 경제적이다.

N이 증가함에 따라 다른 위상이 더 좋을수 있다.

선형배렬은 련결된 많은 마디들이 교환을 통하여 시간공유하는 모선과 크게 차이난다는데 주의하여야 한다.

선형배렬은 서로 다른 사용자들이 폭포의 서로 다른 부분들(련결들)을 동시에 사용할수 있도록 한다.

고리(Ring)

고리는 선형배렬의 두개의 끝마디를 하나의 보충련결로 접속하여 단긴 고리를 형성하게 함으로써 얻어 진다(그림 6-8 ㄴ).

고리는 단방향 또는 쌍방향일수 있다.

고리는 일정한 마디차수 2를 가지므로 균형적이다.

쌍방향고리의 직경은 $N/2$이며 단방향고리의 직경은 $N-1$이다. IBM통표고리는 이 위상을 가지는데 통보문은 자기의 통표와 정합되는 목적지에 도착할 때까지 고리를 따라 돈다.

화음고리(Chordal Ring)

마디의 차수를 2에서 3 또는 4로 증가시켜 각각 그림 6-8 ㄷ)와 그림 6-8 ㄹ)에 제시한 두개의 화음고리를 얻을수 있다. 일반적으로 마디의 차수가 더 높아 지면 더 많은 가

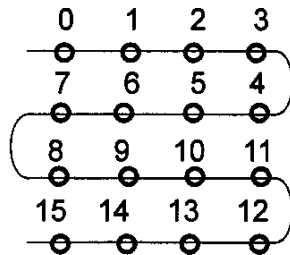
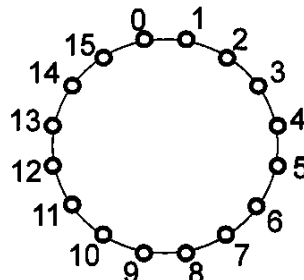
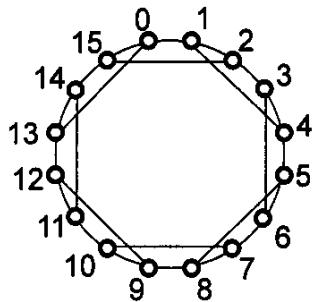
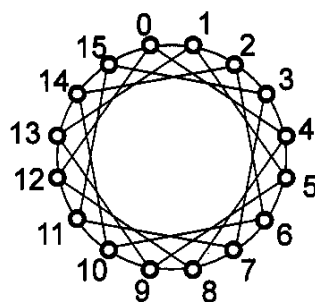
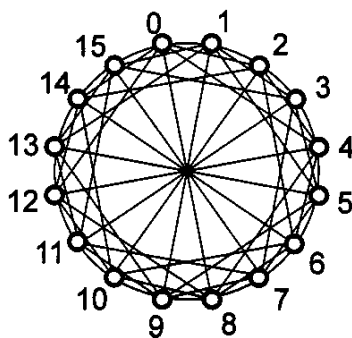
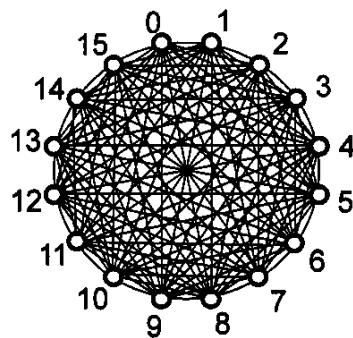



그림 6-8. 마디차수와 접속성의 증가에 따르는 6개의 망위상들

지들이 첨가되며 망의 직경은 더 짧아 진다. 더 높은 마디차수를 가지는 화음망의 실현은 더 많은 련결들을 요구한다. 이 두개의 화음고리를 16개 마디를 가지는 고리와 비교해보면 망직경이 각각 8에서 5로, 8에서 3으로 떨어 짐을 알수 있다.

극단적으로 그림 6—8 ㅂ)의 완전련결망은 가능한 가장 짧은 직경 1을 가지는 반면에 가장 높은 마디차수 N—1을 가진다. 마디차수와 망직경사이의 절충은 주어 진 호상접속망의 확대가능성과 고장허용뿐아니라 가격 대 성능비에도 영향을 준다.

Barrel Shifter

그림 6−8 ㅁ)에 제시한 barrel shifter는 고리의 매 마디로부터 2hop의 정수제곱만한 거리에 떨어 져 있는 마디들에로 련결들을 첨가하여 얻는다.

이것은 어떤 $r=0, 1, 2, \cdots, n-1$에 대하여 $|j-i|=2^r$이면 마디 i는 마디 j에 련결되며 망크기는 $N=2^n$임을 말해 준다.

이 barrel shifter의 마디차수 d는 $2n-1$이며 직경 D는 $n/2$이다.

명백히 barrel shifter의 접속성은 더 낮은 마디차수를 가지는 임의의 화음고리보다 더 높다.

$N=16$일 때 마디차수는 7이고 직경은 2이다.

제시된 모든 위상들은 망크기가 좀 작을 때에만 리용되는 완전련결망보다 더 낮은 복잡성을 가진다.

나무(Tree)

7개의 마디로 된 높이가 3인 2진나무를 그림 6−9 ㄱ)에 제시한다. 2진나무의 마디차수는 3이다. 높이가 k인 2진나무는 $N=2^k-1$개의 마디를 가진다. 2진나무는 매 마디가 두개이상의 나가는 가지를 가지는 다진나무로 확장할수 있다. 일반적으로 m진나무는 $N=m^k-1$개의 마디를 가진다. 잎이 아닌 매 마디(부모마디)는 m개까지의 새끼마디들을 가질수 있다.

별(star)

별(그림 6−9 ㄴ)은 높이가 2이고 가장 높은 마디차수가 $N-1$이며 가장 작은 직경이 2인 다진나무이다.

나무 또는 별형망을 리용하는데서 한가지 문제는 나무의 뿌리나 별형망의 중심마디에서의 병목효과이다. 잎마디들이 다른 마디들과 통신할 때 통신흐름은 뿌리나 중심마디에서 더 무겁다.

그물(Mesh)

2차원 3×3그물망을 그림 6−9 ㄷ)에서 보여 준다.

이것은 Goodyear MPP와 Intel Paragon과 같은 이전의 여러 체계들에서 실현된 대중적인 구조이다. 일반적으로 매개 차원에 n개의 마디를 가지는 k차원그물에서 내부의 마디차수는 2k이며 망직경은 $k(n-1)$이다. 그물은 경계마디들이 내부마디들과 차이나기때문에 균형망이 아니라는데 주의하여야 한다.

Illiac그물

그림 6-9 ㄹ)에 그물의 변종인 Illiac그물을 제시한다. 이 그림은 Illiac망의 축소판이다. 본래의 Illiac망은 마디차수가 4이고 직경이 7인 8×8그물이였다.

일반적으로 $n \times n$ Illiac그물은 직경이 $n-1$인데 이것은 2차원그물의 절반밖에 안된다. 그림 6-9 ㄹ)의 Illiac그물은 위상적으로 그림 6-8 ㄹ)의 차수가 4인 화음고리와 동등하다.

고리면

2차원고리면을 그림 6-9 ㅁ)에 제시한다.

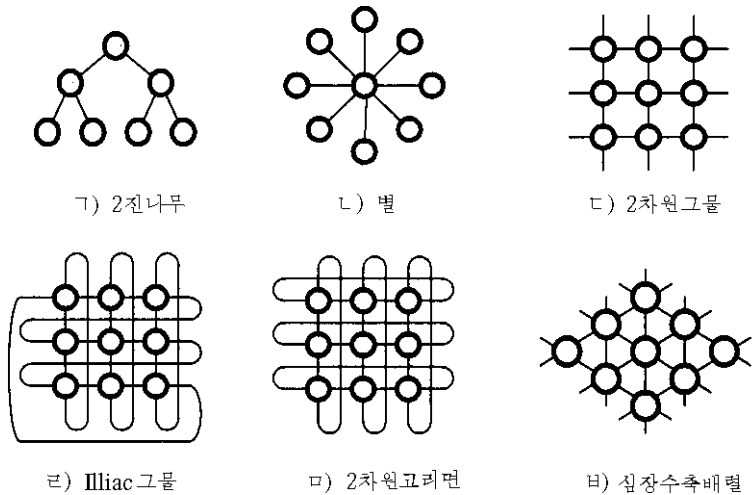



그림 6-9. 망구성을 위한 6개의 보충적인 위상들

이 구조는 접속들을 보충하여 그물로부터 확장된다. 고리면의 직경은 그물직경의 절반이다.

또한 이 위상은 고리와 그물의 우점을 결합하고 있다고 볼수 있다. 그물이나 고리면은 다 더 높은 차원으로 확장할수 있다. 실례로 Gray Research Inc는 자기의 T3D/T3E계렬에서 3차원고리면망을 실현하고 있다.

고리면은 균형적인 위상이므로 그물은 아니다. 보충된 모든 접속들은 고리면의 직경을 줄이고 균형성을 보존하도록 한다.

심장수축배렬(Systolic Array)

이것은 일정한 자료흐름알고리듬들을 VLISI로 실현하기 위하여 설계된 응용지향배렬구조들중의 특정한 종류이다.

그림 6-9 ㅂ)에서 보여 주는것은 행렬 대 행렬곱하기를 위하여 특별히 설계된 심장수축배렬이다. 정적인 심장수축배렬들은 자료스트림들의 다방향흐름과 함께 관흐름된다.

구조는 일단 주어 진 응용에 최량화되면 다른 알고리듬을 효과적으로 실현할수 없다.

굵은 나무(Fat Tree)

나무구조의 뿌리에서 병목현상을 완화하기 위하여 Leiseson은 1985년에 다진 굵은 나무를 도입하였다.

굵은 나무에서 부모마디와 자식마디사이의 련결의 수는 잎마디로부터 뿌리로 가면서 증가한다.

굵은 나무는 뿌리로 가면서 가지가 더 두터워 지는 자연계의 나무와 비슷하다. 굵은 나무구조는 Connection Machine CM5체계에서 실현되였다. CM5의 자료망구축에는 4진굵은 나무가 적합하였다.

CM5에 구속된 굵은 나무는 그림 6-10의 내부마디련결이 보여 주는바와 같이 4개의 자식마디와 2 또는 4개의 부모마디를 가진다.

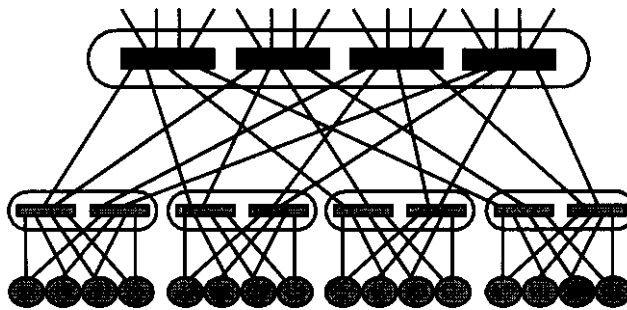



그림 6-10. CM5에서 실현된 4진굵은나무

하이퍼립방체(Hypercube)

형식적으로 하이퍼립방체를 2진n차원**립방체**라고 부른다.

2진이라는것은 립방체의 매 차원(변)이 두개의 마디를 가진다는것이다. 여기서 n을 하이퍼립방체의 **차원수**라고 부른다. 일반적으로 n차원립방체는 n개 차원들을 따라 놓여있는 $N=2^n$개의 마디들로 구성된다.

8개마디로 된 3차원립방체는 그림 6-11 ㄱ)와 같다.

4차원립방체는 그림 6-11 ㄴ)에서 보여 주는바와 같이 2개의 3차원립방체의 대응하는 마디들을 호상접속함으로써 만들어 진다. n차원립방체의 마디차수는 n이며 망직경이다.

사실상 마디차수는 차원수에 따라 선형으로 증가하며 하이퍼립방체구조의 크기확대를 곤난하게 한다.

2진립방체는 1980년대에 연구개발된것으로서 매우 대중화된 구조이다. Intel:PSC/1, :PSC/2, nCUBE, Connection Machine의 CM2기계들은 하이퍼립방체구조로 만들어 졌다. 구조는 빽빽한 접속들을 가진다. 2진나무들, 그물들, 다른 많은 저차원위상들은 하이퍼립방체에 매몰될수 있다.

고차원립방체를 묶는데서의 약한 확대가능성과 곤난성으로 하여 하이퍼립방체구조는 점차 저차원구조로 교체되였다. 실례로 Connection Machine CM5는 CM2에서 실현된 하이퍼립방체대신에 굵은 나무를 선택한다.

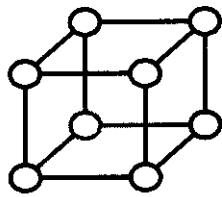
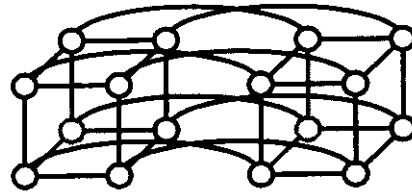
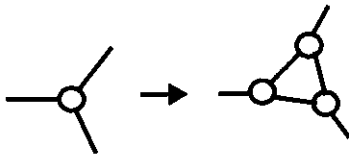
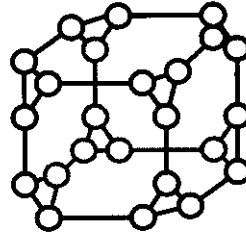



그림 6-11. 2진하이퍼립방체와 립방체접속순환(CCC)

Intel Paragon은 하이퍼립방체적인 먼저것대신에 2차원그물을 선택한다. 리상적인 동등성은 여러 망구조속에서 실현된다.

구조가 계속 존재하기 위한 길은 중요하게 묶기와 효률, 확대가능성, 성능에서의 가격 대 성능절충에 있다.

Wormhole경로정하기와 관흐름망운영으로 하여 정규적인 위상선택이 전보다 그리 중요치 않게 된다.

다시 말하여 구조적인 위상은 호상접속망의 성능을 결정하는데서 그리 예민하지 않거나 그리 중요하지 않게 된다. 우위를 차지하는것은 매체와 교환기술이다.

립방체접속순환(Cube-Connected Cycles)

이 구조는 하이퍼립방체로부터 변경된다. 그림 6-11 ㄷ)에서 보여 주는바와 같이 3차원립방체는 립방체접속순환(3-CCCs)에로 변경된다.

방법은 3차원립방체의 매 구석마디를 절단하고 3개의 마디로 된 하나의 순환으로 교체하는것이다.

일반적으로 $n=2^k$개의 순환마디를 가지는 k차원립방체로부터 k차원립방체접속순환을 만들수 있다. 방법은 k차원하이퍼립방체의 매 정점을 k개의 마디로 된 순환으로 교체하는것이다.

따라서 k차원립방체는 $k\times2^k$마디를 가지는 k차원 CCC로 절환될수 있다. 그림 6-11 ㄹ)에서 보여 준 3차원 CCC는 직경이 6으로써 본래 3차원립방체의 두배이다.

일반적으로 k차원 CCC의 망직경은 2k이다. 하이퍼립방체에 비한 CCC의 우점은 고정된 마디차수 3에 있는데 이것은 바탕으로 되는 하이퍼립방체의 차원과 독립이다.

실례로 64마디 CCC는 4차원립방체의 구석마디들을 4개 마디로 된 순환들로 교체함으로써 형성되는데 $n=6$이고 $k=4$인 경우에 대응한다. CCC는 직경 $2k=8$을 가진다.

그러나 CCC는 6차원립방체의 마디차수 6보다 작은 마디차수 3을 가진다.

이런 면에서 CCC는 만일 어떤 방법으로 지연이 허용된다면 확대가능한 체계를 구성하기 위한 더 좋은 구조로 된다.

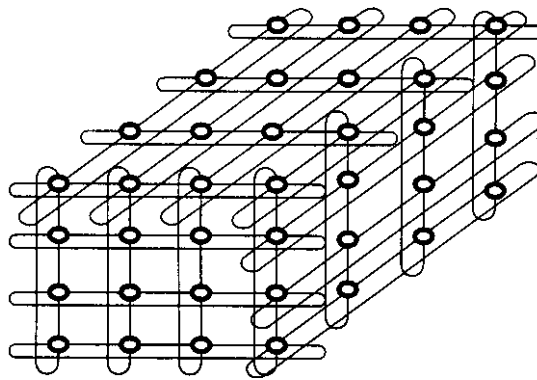



그림 6-12. 4-ary 3차원립방체망(은페된 부분은 보여 주지 않는다.)

k-ary n차원립방체망들

고리와 그물, 고리면들, 2진n차원립방체(하이퍼립방체), 오메가(Omega)망들은 모두 k-ary n차원립방체망계통과 위상적으로 같은 모양이다. 그림 6-12에서 매 차원에 $k=4$개 마디를 가지는 4-ary 3차원립방체망을 보여 준다.

명백성을 위해서 숨은 마디들과 숨은 련결들은 보여 주지 않는다. 이 망에는 $N=k^n (k=n\sqrt{N}, n=\log_k N)$개 마디가 있다. k-ary n차원립방체의 마디는 n개 수자들의 렬 $A=a_0 a_1 \cdots a_n$로 식별할수 있다.

여기서 a_i는 i번째 차원에서 마디의 위치를 나타낸다.

일반적으로 저차원 k-ary n차원립방체들은 n=2 또는 n=3인 고리면으로 불리운다.

고차원 2진 n차원립방체들을 **하이퍼립방체**라고 부른다.

굵은 나무와 k-ary n차원립방체망은 다른 망들을 모형화하는데서 만능적이다.

Dally는 k-ary n차원립방체망의 일부 흥미 있는 속성들을 밝혀 냈다. 대부분의 망들의 가격은 요구되는 교환기들의 수보다는 선의 량에 따라 결정된다. 만일 고정된 선으로 제한하면 더 넓은 통로를 가지는 저차원망들은 좁은 통로를 가지는 고차원망들보다 낮은 지연과 적은 충돌, 더 높은 처리량을 제공한다.

망위상들에 대한 개괄

표 6-1에 망위상들의 중요한 특징들을 개괄한다.

마디차수가 4 또는 그이하의 망들이 보다 합리적이다.

실례로 INMOS Transprinter소편은 2차원그물구조에 4개의 통로에 접속되는 4개의 포구를 가진 극소형처리기이다.

완전접속망들과 별형망들은 다 높은 마디차수에는 대단히 나쁘다. 하이퍼립방체의 마디차수는 $\log_2 N$으로 증가하는데 N이 대단히 크게 되면 역시 나쁘다.

표 6-1 정적접속망들의 위상적속성

망형태	마디차수(d)	망직경(D)	련결수(l)	2등분폭(B)	균형성	망크기
선형배렬	2	$N-1$	$N-1$	1	아니	N개 마디
고리	2	$N/2$	N	2	예	N개 마디
완전접속	$N-1$	1	$N(N-1)/2$	$(N/2)^2$	예	N개 마디
2진나무	3	$2(h-1)$	$N-1$	1	아니	나무높이 $h=[\log_2 N]$
별	$N-1$	2	$N-1$	$N/2$	아니	N개 마디
2차원그물	4	$2(r-1)$	$2N-2r$	R	아니	$r\times r$그물 $(r=\sqrt{N})$
일리아크그물	4	$r-1$	$2N$	$2r$	아니	$(r=\sqrt{N})$인 화음고리
1차원고리면	4	$2[r/2]$	$2N$	$2r$	예	$(r=\sqrt{N})$인 $r\times r$고리면
하이퍼립방체	N	N	$nN/2$	$N/2$	예	$N=2^n$개 마디
CCC	3	2k-1+[k/2]	$3N/3$	$N/(2k)$	예	$K\geqq3$인 순환을 가지는 $N=k2k$개 마디
k-ary n차원립방체	$2n$	$N[k/2]$	nN	$2k^{n-1}$	예	$N=k^n$개 마디

관흐름식wormhole경로정하기에 의해 망직경은 전보다 그리 위험하지 않다. 왜냐하면 임의의 두 마디사이의 하드웨어지연은 경로길이와 거의 독립이기때문이다.

사용된 련결의 수는 망가격에 최대의 영향을 준다. 2등분폭과 통로폭은 망의 대역너비를 결정한다.

망의 균형성은 확대가능성과 경로정하기의 효률을 높이기 위하여 필요하다.

6. 3. 모선과 크로스바 및 다단교환기

동적접속은 여러 통신경로들의 교차점에 전자교환기, 경로기, 집중기(concentrator), 분배기(distributor), 중계기(arbitrator)들을 배치함으로써 실현된다.

동적접속의 3가지 요소는 모선과 크로스바교환기, 다단교환기들이다.

그림 6-1에서 본바와 같이 호상접속들은 현재 200Mbps와 100Gbps사이의 자료전송

속도(대역너비)를 가진다.

6. 3. 1. 다중처리기모선

모선은 본질에 있어서 처리기와 기억모듈, 주변장치들사이 자료처리를 위한 선과 접속기(connector)들의 집합체이다. 체계모선은 처리기와 같은 주장치와 기억기판(memory board)과 같은 종속장치사이의 자료전송에 사용된다. 모선중계론리는 한번에 하나의 요구에 대한 모선호출을 허용하며 따라서 충돌모선이라는 이름을 허용한다.

PCI와 VME, Multibus, MicroChannel, IEEE Futurebus와 같은 많은 표준모선들이 확립되였다.

대부분의 표준모선들은 단일처리기체계를 구성할 때 가장 낮은 가격을 제공한다.

우리는 대규모SMP와 NUMA, DSM기계를 만들기 위한 다중처리기모선과 계층모선들에 초점을 둔다.

이러한 확대가능모선들은 흔히 캐쉬일관성, 고속극소형처리기동기화, 분산작용들의 새치기처리 등을 지원하는 하드웨어로 장비된다.

그림 6-13에 전형적인 다중처리기모선체계의 구조를 보여 준다. 체계모선은 흔히 뒤판이나 중심판에 실현된다.

매 처리기(P) 또는 매 I/O처리기(IOP)는 특별한 종속장치(기억기나 디스크구동기 등)에 대한 접근요구를 생성하는 지배요소이다.

체계모선은 자료경로, 주소선들, 조종선들로 구성되며 모든 끼움식기능기판들의 통신을 위한 공동매체를 제공한다.

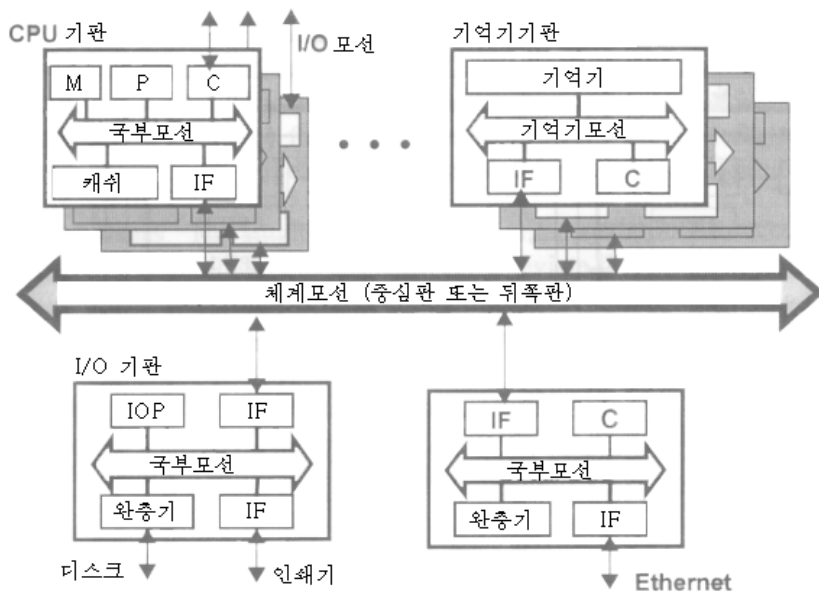



그림 6-13. 체계모선과 기억기모선, 여러가지 기능기판의 국부모선들로 이루어 지는 다중처리기모선

특정화된 대면부론리(IF)와 특수한 기능의 조종기들(C)은 서로 다른 끼움식기판에 사용된다.

국부모선들은 CPU나 I/O, 망대면부기판들에 있다. 기억기기판의 국부모선을 특별하게 **기억기모선**이라고 부른다. 전형적인 I/O모선에는 SCSI 또는 국부디스크와 인쇄기, 다른 호스트기계들에 설치된 주변장치들을 련결하는 I/O통로들이 있다.

다중처리기모선체계를 설계하는데서 중요한 문제에는 모선중계, 새치기처리, 규약변환, 모선다리, 계층모선확장 등이 있다.

종종 snoopy규약들이 5.2.2에서 론의한것과 같은 다중처리기모선에 설치된다.

아래에서 대중적인 SMP봉사에서 개발된 체계모선들중 Sun Microsystems Gigaplane모선과 SGI POWER path-2모선을 고찰한다.

실례 6.1. Sun Microsystems 2.5 Gbps Gigaplane모선호상접속

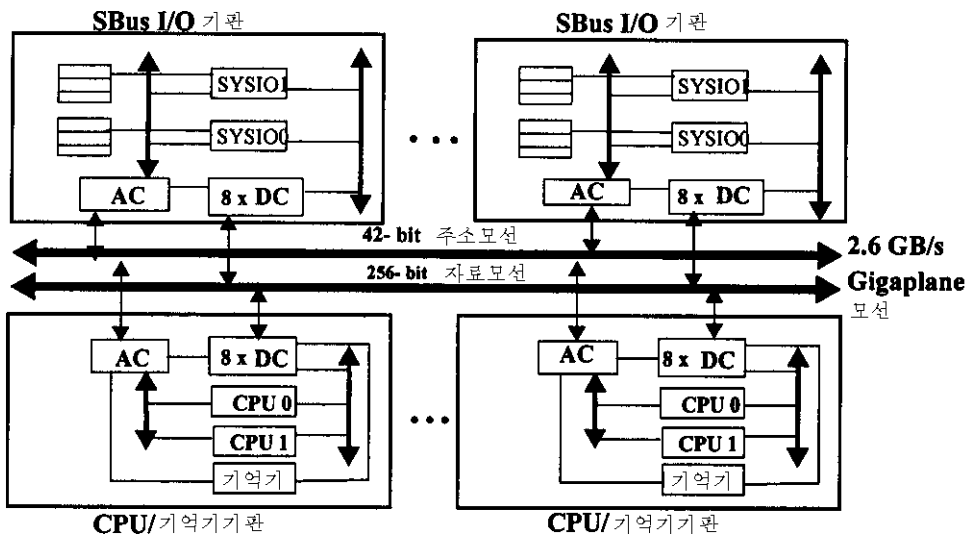



그림 6-14. Sun Enterprise x000 봉사기의 Gigaplane 모선

그림 6-14에서 보여 주는바와 같이 Sun의 Ultra Enterprise Server(SunFire)는 CPU/기억기기판들과 Sbus I/O기판들을 호상접속하기 위하여 Gigaplane이라는 새로운 체계모선을 사용하였다. 그것은 2.5GB/s의 최대대역너비를 보장하며 많은 봉사기들에서 사용되는 Sun의 XDBUS보다 6배 빠르다.

Gigaplane모선의 자료경로는 2TB의 공유기억을 주소화하기 위한 42bit주소선을 가지는 256bit파케트교환으로 설계된다.

모선박자속도는 83.4MHz이다.

2개의 모선주기마다 전용된 64B의 캐쉬행을 가질 때 전체 폭발적인 자료전송속도는 2.6GB/s이다.

처리기들과 기억기, I/O모선들은 직접 Gigaplane에 접속하며 매개 기판은 Gigaplane을 개별적인 단들로 전개하기 위하여 UPA(Ultra port architecture)호상접속을 사용한다.

실례 6.2. Silicon Graphic POWERpath−2 TM Bus

이것은 1.2GB/s의 최대기억기대역너비를 가진 256bit동기모선이다. 36개까지의 MIPS®R1000®처리기들과 16GB의 기억기, 그라픽스, 6개의 I/O모선들을 그림 6−15에서 제시한 POWER path−2모선에 접속할수 있다.

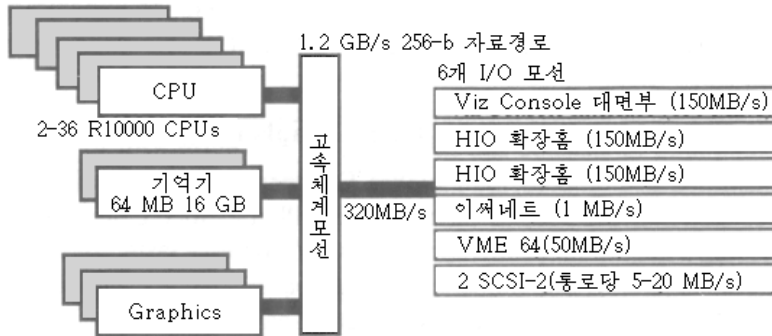



그림 6−15. Silicon Graphic Server의 POWERpath−2TM 모선구조

체계모선은 분산작용규약을 사용한다. 매 CPU기판에 4개의 R1000이 설치된다.

I/O기판당 320MB/s를 가지는 I/O부분체계는 500MB/s까지의 자료전송속도를 얻을수 있다. 고성능그라픽스는 초고속콤퓨터응용들에서 시각화를 지원한다.

많은 전용다중처리기모선들이 IBM대형콤퓨터들과 Digital 및 Hewlett−Packard다중처리기봉사기들에 설치되였다.

IEEE 역시 다중처리기를 위한 VME모선(IEEE 표준 1014−1987), Multibus Ⅱ(IEEE표준 1296−1987), Futurebus(IEEE 표준 896, 1−1991)을 개발하였다.

전기적 및 기계적, 기능적규약들에 대한 서술을 위하여 공개된 표준모선들을 참고해도 된다.

계층모선(Hierarchical Buses)

단일SMP모선은 큰 규모의 체계들에 적응시키는데서 제한된 확대가능성을 가진다. 계층모선구조는 이 문제를 어느 정도로 해소할수 있다.

그림 6−16은 여러개의 다중처리기클라스터들을 호상접속하여 CC−NUMA기계를 만들기 위해 설계된 개념적인 모선계층도이다.

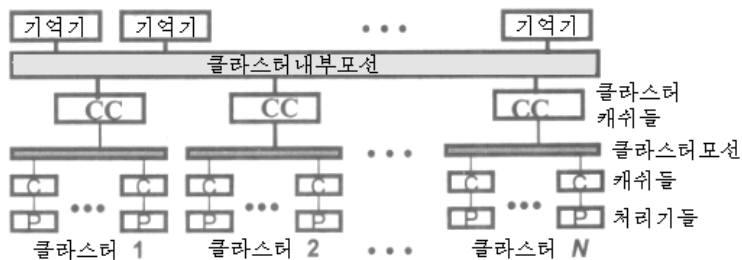



그림 6−16. 다중처리기클라스터들을 접속하여 CC−NUMA기계를 만들기 위한 모선계층도

같은(SMP봉사기와 같은) 클라스터에 속하는 모든 처리기들은 공동클라스터모선에 련결한다. 클라스터캐쉬(CC)는 같은 클라스터의 모든 처리기들이 공유할수 있는 2차캐쉬로 사용될수 있다. 다중처리기클라스터들은 대역적으로 공유된 모든 기억기모듈들을 접속하는 클라스터호상모선을 통하여 서로 통신한다.

여러 수준의 모선들은 모든 전용 및 공유캐쉬들사이의 캐쉬일관성을 유지하고 클라스터들사이를 대면시키기 위한 다리기구를 갖추어야 한다.

IEEE Futurebus는 계층적인 모선체계를 만들기 위하여 특수한 다리들과 캐쉬 및 기억기 agent들, 통보문대면부들, 캐블토막들을 개발하였다.

모선호상접속의 부족점

모선호상접속은 많은 처리기들이 모선을 시간공유하도록 한다. 모선대역너비가 높을 때에도 처리기당 대역너비는 전체 대역너비의 한 부분에 불과하다. 더우기 모선여유가 없기때문에 실패하기 쉽다.

모선은 또한 제한된 확대가능성을 가진다.

이러한 부족점들은 기본적으로 묶기기술과 비용에 의하여 제약된다.

모선들은 흔히 작은 시렁(rack)내로 제한된다. 여러 시렁들을 벗어 나는 계층적인 모선확장에서는 박자비뚤어짐과 대역박자를 없애기 힘들다.

크로스바교환기나 다단망들로 모선구조를 개선함으로써 이러한 부족점을 어느 정도 극복할수 있다.

6. 3. 2. 크로스바교환기

같은 포구당자료경로폭과 같은 수의 접속포구들에 대하여 크로스바교환기망은 훨씬 더 높은 대역너비를 제공할수 있다.

크로스바는 단일단계(single-stage)교환망이다.

전화교환기기판과 같이 교차점의 교환기들은 모든(원천, 목적) 쌍들사이의 동적인 접속을 제공한다. 교차점교환기들은 프로그람의 조종밑에 동적으로 접속(on) 또는 접속해제(off)로 될수 있다.

크로스바교환기를 사용하는 두가지 방법은 다음과 같다.

하나는 균형적인 다중처리기 또는 다중콤퓨터클라스터들사이의 처리기호상통신을 위한것이고 다른것은 SMP봉사기나 벡토르초고속콤퓨터들에서 처리기-기억기호상접근을 위한것이다.

실례 6.3. Digital의 GIGAswitch/FDDI크로스바교환기

그림 6-17에 Digital GIGAswitch/FDDI의 크로스바설계를 제시한다.

이 크로스바는 Alpha워크스테이션과 봉사기들의 여러 FDDI들사이의 클라스터호상접속으로 설계되였다. FDDI완전중복기술(FFDT)을 사용함으로써 매 통로가 100Mbps인 22개까지의 FDDI포구들을 접속할수 있다.

그림 6-17에서 GIGAswitch/FDDI크로스바교환기는 알파워크스테이션/봉사기 farm들

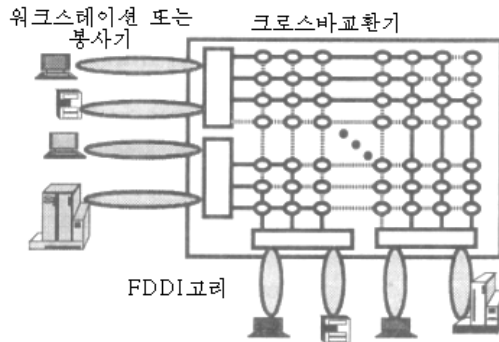



그림 6-17. Alpha 워크스테이션/봉사기를 위한 크로스바교환기 GIGAswitch/FDDI

에 설치되였다.

cut-through경로정하기를 리용하면 지연은 20㎲이하로 작아 진다. 2 또는 4개의 포구선기판들이 22개포구크로스바교환기를 만드는데 사용된다.

전체적으로 GIGAswitch/FDDI는 3.6Gbps까지의 대역너비를 실현하는데 이것은 이써네트보다 360배나 더 높다.

GIGAswitch/FDDI는 Digital의 TruCluster에 사용되였다. GIGAswitch/FDDI는 임의의 FDDI고리접속식체계에서 사용할수 있는 일반적인 망제품이다. 이것은 크로스바설계에 기초한 마디호상통신체계의 좋은 실례로 된다.

일반적으로 크로스바의 복잡성은 N^2으로 증가한다. 여기서 N는 포구의 수이다.

가격상제한으로 하여 앞으로 대단히 큰 규모의 크로스바교환기제작은 기대할수 없다.

처리기호상간의 크로스바는 N개 처리기들사이의 치환을 제공한다. 최근에 Sun Microsystems는 Ultra Enterprise(StarFire) SMP봉사기의 Gigaplane모선을 Gigaplane-xB호상접속으로 개량하였다.

이것은 분할된 주소 및 자료경로들을 가지는 파케트교환구조에 의한 크로스바호상접속이다. 64개까지의 처리기들이 4개의 snoopy주소모선들과 하나의 16×16자료크로스바교환기에 의해 접속된다.

크로스바는 처리기들사이의 점대점자료통신에 사용되며 주소분배는 snoopy모선의 방송경로기들을 처리한다.

SunFile체계는 앞으로 8장에서 학습한다.

크로스바소편설계

크로스바망들은 크로스바교환기소편들로 구성된다.

이 크로스바소편들은 다단망들에서도 사용될수 있다.

아래에서 IBM SP2다중콤퓨터체계의 크로스바소편설계를 고찰한다.

실례 6.4. IBM Vulcan크로스바소편설계

IBM Vulcan크로스바교환기소편설계는 그림 6-18에 제시되였다. 이 교환기는 IBM SP2에서 다단망의 구성요소로 사용되였다. 매 소편은 8개의 입력포구와 8개의 출력포구

를 가진다. 단순화한 wormhole경로정하기를 사용한다. 8×8크로스바는 충돌이 없을 때 40MHz주기마다 8개의 파케트세포(flit라고 한다.)들을 교환기로 통과시킬수 있다.

기억기호출충돌분쟁이 있을 때는 한번에 하나의 교차점교환기만 파케트를 통과시킬 수 있다. 차단된 flit들은 중앙대기렬에서 완충된다.

이 완충은 이전의 교환단계로부터 다음의 flit들을 받기 위하여 입력포구들을 해제한다. 중앙대기렬은 한 박자주기에 한번의 읽기와 한번의 쓰기를 수행할수 있는 이중포구 RAM으로 실현한다.

최대대역너비와 정합시키기 위하여 매 입력포구는 먼저 자기의 FIFO로부터 8개의 flit들을 하나의 덩어리로 만든 다음 전체 64bit덩어리를 한 주기에 중앙대기렬로 쓴다.

처리기−기억기크로스바교환기(Processor−Memory Crossbar Switch)

모선접속된 다중처리기는 모선대역너비에 의해 제약된다. 대단히 많은 처리기들이 공유기억기에 접근할 때 모선은 빈번히 병목으로 된다.

더 좋은 방법은 그림 6−19에서와 같이 처리기호상간의 기억기모선을 크로스바교환기로 교체하는것이다.

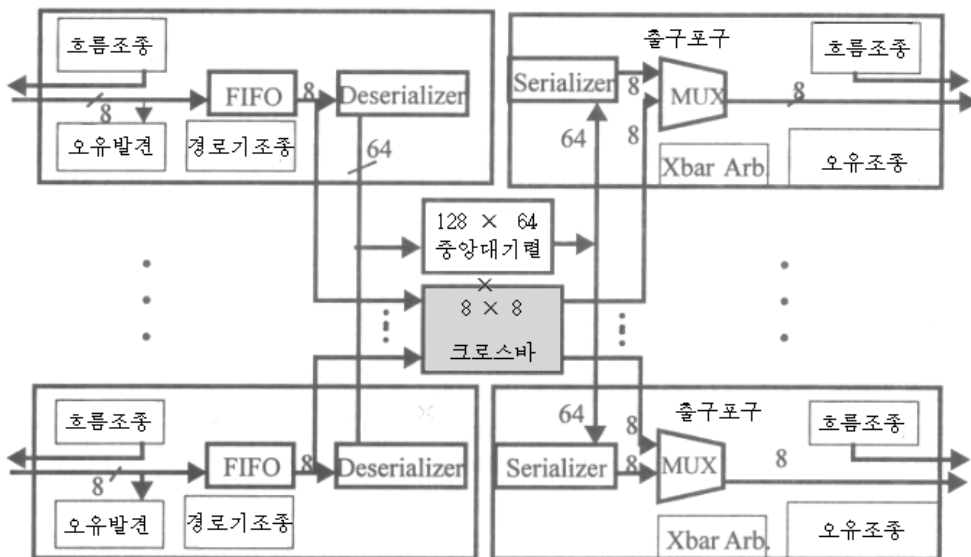



그림 6−18. SP2 HPS를 위한 8×8크로스바(Vulcan)소편설계

이것은 여러 기억기장소들에 대한 기억기에 병렬로 접근할수 있게 한다.

이것은 본질에 있어서 기억기접근망이다. 주요우점은 기억기대역너비 또는 처리기와 공유기억기들사이의 자료전송속도가 뚜렷하게 증가한다는데 있다.

매 기억기모듈에는 한번에 하나의 처리기만 접근할수 있다. 여러개의 요청이 하나의 모듈에서 충돌하면 크로스바는 그 충돌을 해소하여야 한다.

매 교차점교환기의 동작은 충돌모선의 경우와 매우 비슷하다.

그러나 처리기는 서로 다른 기억기모듈들에 접근하기 위한 주소렬을 관흐름형식으로 또는 동시에 생성할수 있다.

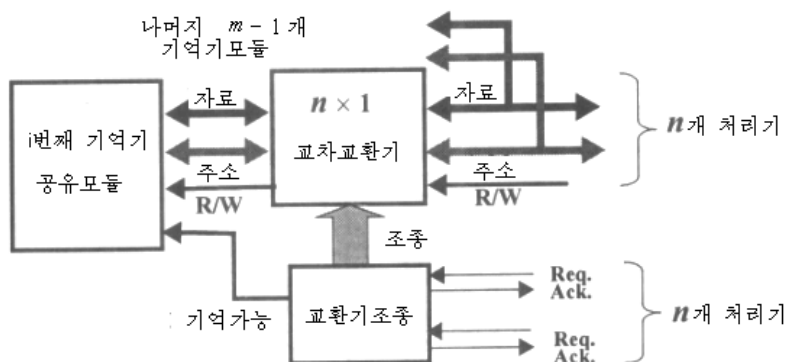



그림 6-19. *n* 개처리기와 *m* 개의 기억기모듈사이의 n×m 크로스바교환기부분체계

일반적인 SMP봉사기들에서는 거의 모든 모듈들이 처리기와 기억기장소사이에 모선접속을 리용한다. 최근에 IBM과 Sun Microsystems는 다른 SMP판매자들로부터 확대가능한 SMD봉사기의 크로스바교환기호상접속에 대한 의견을 받았다.

6. 3. 3. 다단호상접속망

큰 교환망을 구성하려면 단일단크로스바를 여러 단으로 확장해야 하는데 이것을 흔히 **다단호상접속망**(MIN)이라고 부른다.

MIMD 및 SIMD콤퓨터들은 MIN을 사용하였다. 매단에 여러개의 교환기모듈들이 사용된다. 린접한 단들의 교환기들사이에서는 고정된 단호상접속이 리용된다. 교환기접속은 입력들과 출력들사이의 목적하는 접속형태를 확립하기 위하여 동적으로 접속 또는 접속해제될수 있다.

교환모듈

n×*n*교환기모듈은 *n*개의 입력와 *n*개의 출력을 가진다. 2진교환기는 그림 6-20 ㄱ)에서 보여 주는바와 같이 2×2교환기모듈에 대응한다. 매 입력은 임의의 출력포구에 접속될수 있다. 그러나 1대 1 또는 1대 n넘기기만이 허용되며 n대 1넘기기는 출력충돌때문에 허용되지 않는다.

오메가망

일반적으로 *n*×*n*크로스바교환기는 *n*!개의 치환접속을 실현할수 있다. 서로 다른 종류의 MIN들은 사용되는 교환기모듈들과 단호상접속(ISC)형태들에서 서로 다르다. 자주 사용되는 ISC형태에는 완전혼합, butterfly, 다통로혼합, 크로스바, 립방체접속 등이 있다.

그림 6-20 ㄱ)에 오메가망을 구성하는데 사용된 2×2교환기의 가능한 4개의 접속을 제시한다.

2×2교환기 8×8오메가망을 그림 6-20 ㄴ)에 제시한다.

오메가망은 일리노이즈대학의 Cendar다중처리기에서 실현되였다.

Cray Y/MP 벡토르다중처리기와 IBM SP2다중콤퓨터의 MIN설계실례를 보자.

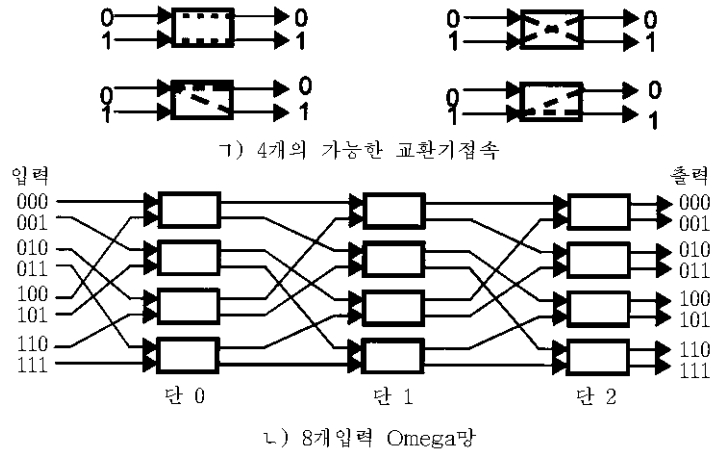



그림 6-20. 2×2 교환기들과 완전혼합에 의한 8×8 오메가망구성

실례 6.5. IBM 고성능교환기(IBM High Performance Switch ,HPS)

마디의 모든 교환기하드웨어들과 교환기프레임들은 고성능교환기(HPS)를 형성한다. 매 프레임은 16개 통로교환기기판으로 접속된 16개의 프로쎄스마디들(N0부터 N15까지)로 구성된다.

8개의 프레임은 교환기기판들의 여유단에 의해 호상접속된다. 매 얇은 선은 8bit쌍방향련결을 나타낸다. 두터운 선은 4개의 8bit련결들을 나타낸다.

매 교환기기판에는 2개의 단이 있다.

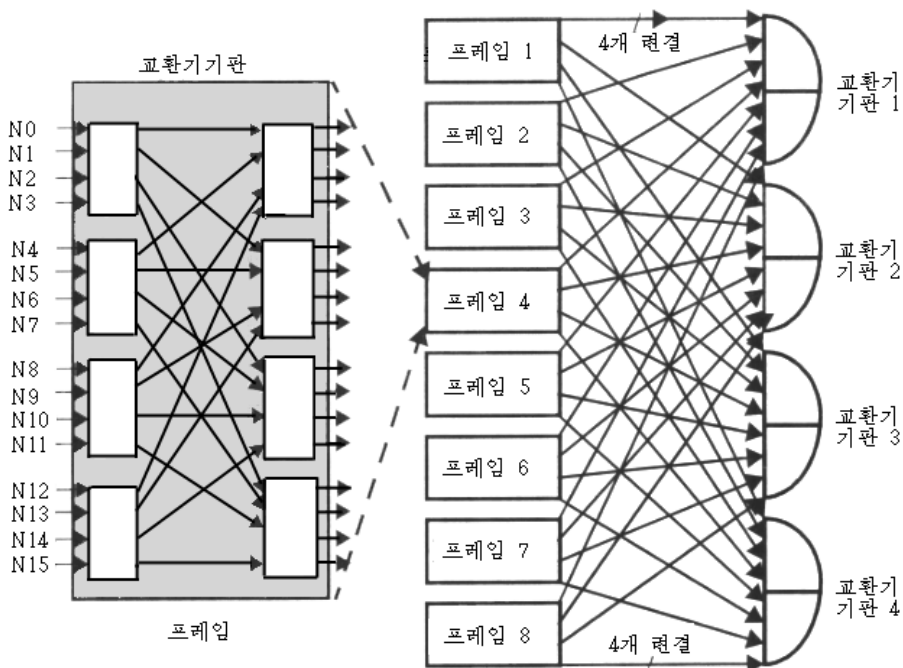



그림 6-21. 4단계의 16통로교환기로 된 IBM SP2의 128통로고성능교환기

총체적으로 이 MIN은 4개의 교환기단을 가진다. HPS는 40MHz의 박자로 동작하는 완충식wormhole경로정하기를 사용하는 파케트교환의 다단오메가망이다.

128개의 마디들을 접속하는 실례 HPS를 그림 6-21에 제시한다. 매 교환기기판은 두개 단의 교환기소편을 가진다. 교환기기판의 왼쪽 련결들은 프레임안의 마디들을 접속하는데 사용된다. 오른쪽 련결들은 프레임호상접속을 제공한다. 매 련결은 8bit이며 쌍방향이다.

통보문들은 파케트들로 분할되고 파케트들은 차례로 8bit의 흐름조종수자(flit)들로 분할된다.

충돌이 있을 때 하나의 flit가 HPS의 한개 단(실례로 하나의 교환기소편)을 통과하는데 5박자 또는 125ns 걸린다. 따라서 충돌이 없을 때 HPS의 하드웨어지연은 875ns로써 작다.

응용의 프로쎄스들에 의한 실제적인 지연은 더 높다. 하나의 프로쎄스가 하나의 빈 통보문을 다른 프로쎄스에로 전송하는데는 적어도 40㎲가 걸린다.

이 통보문넘기기지연은 쏘프트웨어부가처리때문에 생긴다. 마디쌍사이에서 HPS는 포구당 40MB/s의 대역너비를 제공한다.

임의의 고성능클라스터체계를 설계하는데서 2개의 중요한 문제가 해결되여야 한다. 모든 마디들에서 박자가 동기화되는것이 좋다. 례를 들면 2개의 마디가 같은 시각에 Unix함수 gettimeofday()를 호출하면 두개의 호출은 같은 값을 되돌린다.

물론 완전동기화는 불가능하다.

어떻게 하면 박자들을 가능한껏 치밀하게(실례로 몇마이크로초인 작은 편차내에서) 맞추겠는가 하는것이 박자동기화문제이다.

두번째 문제는 긴 선문제이다.

클라스터마디들은 MPP마디보다 더 복잡하므로 일반적인 MPP와 같이 치밀하게 묶을수 없다.

마디사이의 통신선들은 보통 1m보다 길다. 선의 길이는 지연과 대역너비에 다 나쁜 영향을 준다. SP2는 동기통신망을 사용하여 박자동기화문제를 해결한다. 전체 HPS는 보통의 40MHz진동자로 박자동기화된다.

동기망은 매 마디가 다른 마디들의 박자와 동기되는 국부 time-of-day박자를 유지할수 있게 한다.

IBM은 동기화된 마디의 박자들을 몇마이크로초내에 동조시키는 worm이라는 프로그람을 개발하였다.

실례 6.6. Cray Y/MP다단망

이 망은 그림 6-22에서 보여 주는바와 같이 4×4 및 8×8크로스바교환기들, 1×8역다중화기들로 구성된다.

망은 8개의 벡토르처리기들과 256개의 기억기장소사이의 자료흐름을 지원하도록 설계된다.

망은 8개의 처리기들에 의한 기억기접근충돌을 피할수 있다.

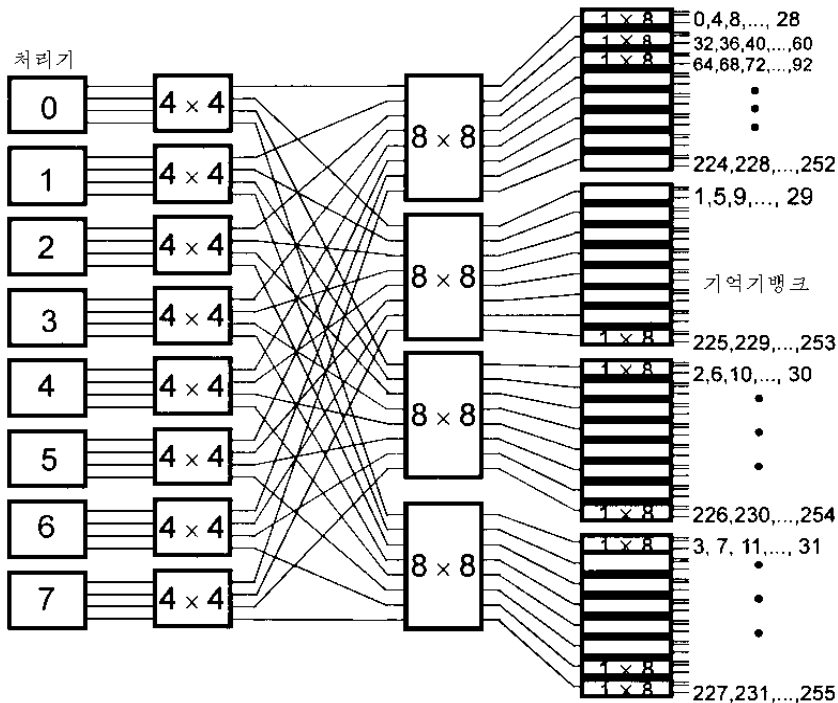



그림 6-22. Cray Y-MP/816의 처리기-기억호상접속을 위한 다단크로스바망

6. 3. 4. 교환호상접속들의 비교

아래에서는 확대가능한 가동환경을 고찰한다. 클라스터체계들에서 동적인 체계호상접속을 실현하기 위한 체계모선들, 다단망들, 크로스바교환기들에 대한 장치요구와 성능을 비교한다. 표 6-2에 교환호상접속의 가선/교환복잡성, 처리기당 대역너비, 집합대역너비를 개괄한다.

하드웨어복잡성

모선호상접속은 제한된 가선 및 교환복잡성으로 하여 세가지중에서 가장 가격이 낮다. 가선복잡성은 기본적으로 자료경로폭과 모선설계의 주소폭에 의하여 규정된다.

256개 자료선과 42개 주소선들은 현재의 모선들의 기술상태를 보여 준다. 주소선들은 256bit Futurebus의 64개의 공유된 주소/자료선들과 같이 자료경로의 부분으로서 은폐시킬수 있다.

모선교환복잡성은 모선의 접속분기의 수 n에 의하여 결정되며 체계모선에 련결되는 적은 수의 처리기들과 기억기 및 I/O기판들에 대해서도 제약된다.

W를 모선의 자료경로폭이라고 하자.

모선호상접속의 하드웨어복잡성은 표 6-2에서 $O(n+w)$로 보여 주는바와 같이 n과 w에 관하여 선형으로 증가한다.

크로스바교환기는 그것의 장치복잡성이 $n^2 w$에 따라서 증가하기때문에 제작비용이 가장 비싸다.

표 6-2 교환호상접속의 복잡성과 대역너비

호상접속의 특징	체계모선	다단망	크로스바교환기
하드웨어복잡성	O(n+w)	O(nwlog$_k$n)	O(n^{2}w)
처리기장치당 대역너비	O(wf/n)부터 O(wf)	O(wf)	O(wf)
알려진 집합대역너비	SunFire봉사기의 Gigaplane모선에 대하여 2.67GB/s	IBM SP2의 512개 마디에 대하여 10.24GB/s	Digital의 GiGAswitch에 대하여 3.4GB/s

여기서 n^2은 크로스바교환기의 교차점의 수에 대응하며 n는 교차점교환기설계의 통로폭이다.

같은 자료경로폭에 대하여 $n\times n$크로스바교환기는 모선호상접속보다 거의 n^2배 더 비싸다.

N개의 입력다단망은 O[($n$log$_k$$n$)w]의 장치복잡성을 가진다.

여기서 $n\log_k n$은 $k\times k$교환기들의 구성요소로 사용된다고 할 때 사용된 교환기의 수에 대응된다. 여기서 w는 MIN설계에서의 련결폭이다.

이 복잡성은 모선들의 두 극단과 크로스바교환기들사이에 놓인다.

MIN은 같은 통로폭에 대하여 크로스바교환기보다 비용이 $n/\log_k n$배 적게 든다는것을 알수 있다.

처리기당 대역너비(Per-Processor Bandwidth)

SMP봉사기에서 모선은 n개의 처리기에 의해 시간공유된다. 그러므로 n개의 처리기는 모선대역너비를 놓고 경쟁한다. 같은 박자속도 f를 가정할 때 매개 단위자료전송에 3가지 호상접속모두에서 1주기만 걸린다. 처리기당 대역너비는 O(wf/n)과 O(wf)의 범위내에 있다. MIN과 크로스바는 다 더 넓은 처리기당 대역너비를 가지며 함수 O(wf)에 따라 선형으로 변한다.

이것은 명백하게 모선구조에서의 병목효과를 가리킨다. MIN에서 하나의 자료쪼각(piece)을 여러 교환기단을 거쳐 전송하는데 많은 박자가 필요하지만 같은 박자속도 f를 가질 때조차도 모선은 하나의 단위자료쪼각을 전송하는데 더 적은 박자(1 또는 2주기)를 요구한다.

그리하여 실제적인 모선당 대역너비는 MIN보다 그리 낮지는 않다. 모든 경우에 크로스바는 짧은 지연(1 또는 2개 주기)과 입력포구와 출력포구사이의 충돌 없는 접속으로 하여 가장 높은 처리기당 대역너비를 가진다.

집합대역너비(Aggregate Bandwidth)

표 6-2의 마지막행에서 시장에서 구입할수 있는 3가지 대표적인 모선, MIN, 크로스바호상접속의 집합대역너비들을 볼수 있다.

여기서 Gigaplane모선은 2.67B/s=21.36Gbps의 집합대역너비를 가진다.

N=24개의 처리기들이 대역너비를 공유한다고 할 때 처리기당 대역너비는

21.36/24=0.89Gbps로 낮다.

다른 한편 GIGAswitch 크로스바는 3.4Gbps의 집합대역너비를 가진다.

이것은 GIGAswitch의 처리기당 대역너비가 Gigahplane모선보다 거의 3.8배 더 높다는 것을 의미한다.

3가지중에서 IBM HPS는 *n*=512포구보다 훨씬 더 큰 구성을 가지며 결과 집합대역너비가 10.24GBps=8192Gbps로 된다. 이러한 원인으로부터 MIN은 모선이나 크로스바호상접속보다 더 확대가능하다고 결론할수 있다.

교환기선택

상업체계들중에서 40개이하의 처리기들을 가지는 SMP체계에는 모선이 아직 가장 비용이 효과적인 체계호상접속이라는것은 의심할바 없다.

크로스바호상접속은 성능이 가격에 비하여 높든가 또는 체계가 좀 작을 때 즉 16개이하의 처리기를 가질 때 선택될수 있다.

이것은 Sun의 StarFire SMP봉사기가 4개의 주소모선과 64개 처리기들사이의 16×16 크로스바교환기호상접속으로 구성되였다는 사실로부터 확증할수 있다. MIN의 주요우월성은 모듈구성에 의한 확대가능성에 있다. 그러나 지연은 망의 단계의 수 log n에 따라 증가한다. 가선 및 교환복잡성의 증가로 인한 가격은 상용망들에서 큰 MIN들을 구성하는데 또 다른 제한으로 된다.

때문에 상업SP2체계는 최대크기가 128개 마디로 된다.

앞으로 MPP나 클라스터체계들을 구성하는데서 점대점위상들이 보다 융통성 있고 정규적으로 구조화된 망들보다 더 확대가능할것이다.

광학적호상접속과 수자신호처리, 극소전자공학의 발전과 함께 대규모MIN 또는 크로스바망들은 더 큰 다중처리기들 또는 다중콤퓨터들에 널리 사용되게 될것이다. 1997년에 개발된 기술에 기초한 64×64크로스바와 512개 입력MIN은 상업체계들에서 구성되였다.

6. 4. 기가비트망기술

기가비트망기술은 매체공유와 점대점, 교환망들에 사용되였다.

아래에서 이 기가비트망기술의 기능적능력과 구조적인 접속, 응용을 특징 짓는다.

6. 4. 1. 빛섬유통로와 FDDI고리

먼저 통로와 망을 구분하자.

다음 빛섬유통로와 FDDI고리들을 고찰한다.

통로와 망

이것들은 처리기들사이의 자료통신이나 처리기들과 주변장치들사이의 통신을 위한 두가지 기본적인 형태이다.

통로는 통신장치들사이의 직접적 또는 교환에 의한 점대점련결을 제공한다. 통로는 전형적으로 하드웨어집약적이며 자료를 높은 속도로 낮은 부가처리를 가지고 넘기기한다.

통로는 미리 정의된 주소로 몇개의 장치들과만 동작한다.

HiPPI, IBM, SCSI는 모두 잘 정의된 자료통로표준들을 가진다.

대비적으로 망은 분산된 마디들(워크스테이션들이나 파일봉사기들, 주변장치들)의 집합체이며 이러한 마디들사이의 호상접속을 지원하는 자체의 규약을 가진다.

망은 쏘프트웨어집약적인것으로 하여 상대적으로 높은 부가처리를 가지며 결과 통로보다 더 느리다. 망들은 예상할수 없는 접속환경에서 동작하므로 통로보다 더 넓은 범위의 과제들을 처리할수 있다.

잘 확립된 망표준에는 IEEE 802와 TCP/IP, ATM규약들이 있다.

빛섬유통로(Fiber Channel)

ANSI X3T11는 통로들의 집적화된 모임으로서의 빛섬유통로(FC)와 망화(networking), 기억, 워크스테이션들과 대형콤퓨터들, 초고속콤퓨터들, 기억장치들, 현시장치들사이의 자료전송을 위한 망표준을 서술하였다. FC표준은 대량정보를 초고속으로 전송하여야 할 필요성을 설명하였다.

이 표준의 목표는 망화와 기억, 자료전송을 위한 하나의 표준을 제공하여 현재 통로와 망의 다양성을 지원하여야 하는 부담으로부터 체계제작자들을 해방하는것이다.

빛섬유통로는 가장 좋은 통신통로와 통신망수단들을 통로사용자들과 망사용자들의 요구에 맞는 새로운 I/O대면부와 결합시키는것을 목적으로 하고 있다.

서로 다른 판매자들의 제품들사이의 호상운영성보증을 위하여 표준을 평가하는 빛섬유통로체계창시(Fiber Channel Systems Initiative, FCSI)라고 하는 협회가 Hewlett-Packard와 IBM, Sun Microsystems에 의해 형성되였다.

빛섬유통로기술

빛섬유통로는 공유매체이다.

이것은 교환기술로도 구성될수 있다.

현재 빛섬유통로는 100부터 133, 200, 400, 800Mbps범위의 속도로 동작한다.

FCSI판매자들은 앞으로 1, 2, 4Gbps라는 더 높은 속도를 달성하여야 한다. 빛섬유통로는 점대점과 고리, 교환식별형의 접속을 지원하는데 사용되였으며 그외에 LAN응용들에서 의뢰기-봉사기해법 또는 집선기(hub)해법도 제공한다.

빛섬유통로는 50m까지 STP동선을 사용하여 100Mbps로 동작하며 10km까지는 한파모습빛섬유를 사용하여 동작한다.

다중방식빛섬유를 가진 빛섬유통로 LAN은 2km까지 200Mbps의 속도를 보장한다. 현재의 높은 쏘프트웨어부가처리는 망련결(optical link)을 사용하는 대부분의 실현에서 빛섬유통로의 최종적인 성능을 255Mbps이하로 제한한다. 빛섬유통로의 유용성은 일부 기가비트 LAN들(6.4.2에서 고찰한 기가비트이써네트)이 빛섬유통로기술에 기초하고 있다는 사실로부터 확증할수 있다.

5층 FC표준

빛섬유통로구조는 표 6-3에서 개괄한바와 같이 5개의 표준층들로 구성된다.

이 층들은 물리매체와 전송속도들(FC-0), 자료의 부호화와 복호화방법(FC-1), 프레임화규약과 흐름조종(FC-2), 공통봉사와 형태선택(FC-3), 웃층규약과 여러가지 자료통로들, 망표준들에 대한 응용대면부(FC-4)를 정의한다.

IBM통로는 Escon(enterprise systems connection)대면부들을 참조한다.

표 6-3 빛섬유통로의 5층 표준들

표준	자료통로들				망규약들			OSI층
FC-4	HiPPI	IBM	SCSI		IEEE802	TCP/IP	AIM	자료 련결층
FC-3	공동봉사(common servises)							
FC-2	프레임만들기/흐름조종/봉사클라스들							
FC-1	부호화/복호화 8B/ 10B							물리층
FC-0	100 Mbps	200 Mbps	400 Mbps	800 Mbps	미래의 더 높은 속도			

FC-3층은 FC접속과 회선교환으로부터 프레임교환에로의 병행, 고정비트속도동시성(constant-bit-rate isochronons), 혼합전송의 4가지 명백한 봉사부류들을 특징 짓는다.

아래 2개의 층은 OSI의 물리층에 대응하며 웃 3개의 층은 OSI의 자료련결층과 류사하다. 아래 3개의 층을 통채로 빛섬유통로물리표준(FCPH)이라고 부른다. 빛섬유통로의 우월성은 단일련결우로 통로와 망의 규약들을 동시에 전송하는 융통성이 있다는 것이다.

이것은 통로와 망자료통신모두에 대한 만능적인 대면부를 제공한다. 이것은 FDDI, 직렬 HiDDI, SCSI, IPI(intelligent peripheral interface), IP, IEEE802.2 등으로 함께 동작할수 있다.

빛섬유통로위상

망위상에서의 융통성은 빛섬유통로의 주요한 문제이다.

이것은 점대점, 중계고리, 교환식구조접속들을 지원한다.

- 점대점접속(그림 6-23 ㄱ)은 콤퓨터를 다른 콤퓨터에 또는 콤퓨터를 세가지 위상의 가능한 가장 높은 대역너비를 가지는 디스크에 접속할수 있다.
- 중계고리 (그림 6-23 ㄴ)는 통표고리에서 126개까지의 장치를 접속한다. 이것은 많은 기억장치들의 호상접속에 좋다. 리용가능한 대역너비는 모든 장치들사이에서 공유된다. 이 고리의 우월성은 낮은 가격에 있는데 그것은 교환기가 필요 없기때문이다.
- 교환식구조위상은 가장 큰 처리량을 준다. 서로 다른 속도의 많은 장치들이 중앙빛섬유교환기에 접속될수 있다.

완충식wormhole경로정하기가 사용된다.

8×8크로스바는 충돌이 없으면 매 40MHz주기에 8개의 파케트세포들(flit라고 한다.)을

교환기로 통과시킨다.

분쟁충돌이 있을 때는 한번에 한개의 교차점교환기만이 동작할수 있다. 차단된 flit들은 중앙대기렬에서 완충된다.

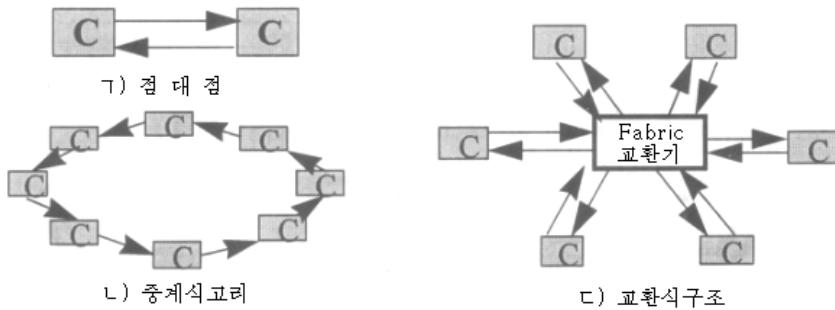



그림 6-23. 빛섬유통로의 3가지 호상접속위상들

이 완충은 이전의 교환기단(stage)으로부터 련속적인 flit들을 받기 위하여 입력포구들을 해제시킨다.

중앙대기렬은 한 박자주기에 하나의 읽기와 하나의 쓰기를 수행할수 있는 2중포구(dual-port)RAM으로 실현된다.

최대대역너비를 맞추기 위하여 매개 입력포구는 먼저 자기의 FIFO로부터 8개의 flit들을 하나의 덩어리로 만들어 전체 64bit덩어리를 1주기의 중심대기렬에 쓴다.

FDDI

Digital Equipment는 매체공유FDDI(fiber distributed data interface)기술을 개발하였다.

FDDI는 워크스테이션(워크스테이션)들사이의 100~200Mbps전송을 제공하기 위하여 쌍방향빛섬유통표고리를 사용한다.

반대방향으로 회전하는 2개의 2중고리는 확실성을 담보하기 위한 여분의 경로들을 제공한다.

이것은 동선에서는 100m, 다중방식빛섬유에서는 2km, 한파모습빛섬유에서는 60km까지 많은 장치들을 호상접속하는 능력을 가진다.

2중련결된 다중방식빛섬유 FDDI고리들은 반복기나 다리들을 사용함이 없이 200km까지 확장할수 있다. 이것은 LAN과 MAN응용들에서 FDDI고리들을 사용할수 있게 한다. FDDI고리들은 고장허용(fault-tolerant)조작들에서도 우월하다. FDDI집중기들은 망을 실패와 분리시킴으로써 믿음성 있게 만든다. 사명이 중요한 봉사기들은 더 높은 고장허용을 위하여 2개의 집중기에 접속할수 있다.

100Mbps기간 FDDI망을 그림 6-24에 보여 준다. FDDI고리들을 많은 탁상콤퓨터들에 접속하는 이써네트집선기에 접속하기 위하여 특수한 경로기들을 사용한다.

FDDI고리는 흔히 보충과 제거를 요구하는 환경이나 임의의 망분리를 일으키지 않고 련결된 호스트나 장치들을 옮길 때에 사용한다.

통표고리 FDDI는 매체공유응용들에서 대중적이다. FDDI는 워크스테이션클라스터들을 지원하기 위하여 더 높은 융통성을 제공하도록 개량될수 있다.

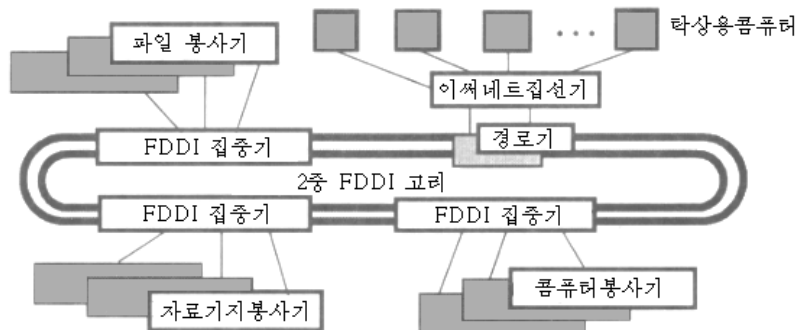



그림 6-24. 기간망으로서의 2중 FDDI 고리들

FDDI의 약점은 FDDI가 비동기적으로만 동작하는것으로 하여 다매체통신흐름지원에서는 무능하다는것이다.

이것은 ATM기술과의 경쟁에서 FDDI를 약화시킨다.

그래서 시간에 예민한 통신흐름을 잘 처리하는 동기FDDI제품들이 출현하였다.

이것은 FDDI가 앞날의 응용들에서 자기의 사용그룹중 일부를 보존할수 있게 한다.

Digital의 FDDI기술은 충분히 중복된 방식으로 FDDI를 사용할수 있게 하며 또 FDDI의 경쟁성을 높여 준다.

6. 4. 2. 고속이써네트와 기가비트이써네트

이 부분에서는 3개의 이써네트세대들을 고찰한다. 그다음 대학환경들에서의 학문적인 연구과제의 경과인 Myrinet를 소개한다.

표 6-4 이써네트개발의 3개 세대

세대	이써네트 10BaseT	고속이써네트 100BaseT	기가비트이써네트
소개된 년도	1982	1994	1997
대역너비(속도)	10 Mbps	100 Mbps	1Gbps
UTP쌍꼬임선	100m	100m	25-100m
STP/동축케블	500m	100m	25-100m
다중방식빛섬유	2km	반중복에서 412, 완전중복에서 2km	500m
한파모습빛섬유	25km	20km	2km
주요응용들	파일공유인쇄기공유	작업그룹콤퓨터의뢰기-봉사기구조, 큰 자료기지접근	큰 화상파일, 다매체, 인트라네트, 인터네트, 자료창고

이써네트세대다중콤퓨터클라스터들이나 인터네트응용들에서 오늘날의 대역너비요구를 더는 충족시킬수 없다.

1994년에 두개의 100Mbps고속이써네트판본(100BaseT와 100VGAnyLAN)이 개발되였다.

1982년에 처음으로 등장한 10Mbps이써네트는 다중콤퓨터클라스 혹은 인터네트에서 오늘의 대역너비요구를 충분히 만족시키지 못한다. 최근 1997년에 IEEE802.3연구집단그룹은 1기가비트이써네트의 유용성을 발표하였다. 이써네트기술의 발전을 속도(대역너비)와 케블길이, 전형적인 응용의 견지에서 표 6-4에 요약하여 제시하였다.

케블거리(Cabling Distance)

이써네트의 최대거리는 사용된 케블기술에 의존하는데 25km까지 보장한다. 기가비트이써네트에서는 망거리가 25m부터 2km까지 범위로 줄어 든다. 표 6-4는 쌍꼬임선(카테고리 5 UTP)으로부터 STP동축케블과 한파모습빛섬유까지 뚜렷한 거리증가를 보여 준다.

기가비트이써네트는 더 큰 대역너비를 요구하는 구내 또는 건물을 목표로 할것이다. 100Base-T는 동선 100m, 한파모습빛섬유 20km, 완전중복다중방식빛섬유 2km에서 100 Mbps로 동작한다.

또 다른 고속이써네트기술은 100VG-AnyLAN인데 쌍꼬임선 100~150m와 빛섬유케블 4km에서 100Mbps의 속도를 제공한다.

이써네트는 대체로 모선 또는 병형위상이며 한편 고속이써네트는 대체로 별형위상을 지원한다. 투자보호를 위하여 기가비트이써네트는 이전의 이써네트세대들에 망하부구조, 관리, 응용들의 변화를 요구하지 않는다.

기가비트이써네트이동(Gigabit이써네트 Migration)

호상운영성(interoperability)과 뒤쪽호환성(backward compatibility)은 기가비트이써네트의 2개의 주요한 수단이다.

이것은 오래동안 이써네트에서 사용된 CSMA/CD접근규약을 유지한다. 빛섬유케블은 초기의 포구당 가격이 좀 높다. 규소(silicon)기술과 수자신호처리의 발전은 앞으로의 Category 5 UTP가선우에서 기가비트망의 경제적인 지원을 가능하게 할것이다.

실례 6.7. 기가비트이써네트 LAN의 기간

그림 6-25에서는 교환식고속이써네트기간을 어떻게 기가비트기간 LAN으로 갱신하는가를 보여 준다. 고성능봉사기 farm들은 기가비트인터네트 NIC들을 가지고 이 기가비트기간에 직접 접속될수 있다.

이 갱신은 인터네트사용자들을 위한 다중봉사기클라스터의 처리량을 증가시킨다.

4가지 이써네트갱신계획은 다음과 같다.

- 응용들과 파일봉사기들에 대한 고속접근을 달성하기 위하여 교환기들을 봉사기련결로 갱신하여야 한다.
- 100 과 1Gbps 교환기들사이의 1Gbps 관흐름을 실현하기 위하여서는 교환기 대 교환기련결들을 갱신하는것이다.

- 교환식이써네트기간을 기가비트이써네트교환기 또는 반복기를 가지는 집체(aggregate) 고속이써네트교환기로 갱신하는것이다.

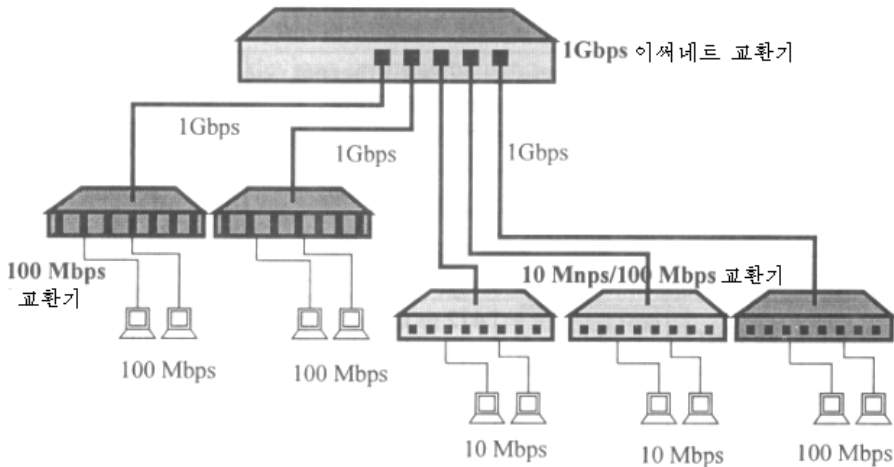



그림 6-25. 기가비트이써네트 LAN기간의 구조

- 공유 FDDI기간을 기가비트이써네트교환기들, 반복지들을 가지는 설치(install) FDDI교환기 또는 이써네트 대 FDDi교환기들/경로기들로 갱신하는것이다.

기가비트이써네트표준 IEEE802.3z는 1978년에 완성되였다. 모든 이써네트세대들은 MAC(media access control)규약 CSMA/CD를 사용한다.

물리층에서는 빛섬유통로와 쌍끄임기술들이 케블련결대면부들에 리용된다. 여러 판매자(Vendor)들이 이미 기가비트이써네트교환기들을 생산하였다. 실례로 Plaintree Systems는 16개의 기가비트이써네트포구들 또는 64개 FDDI SAS포구들, 128개 100Base-TX/FX포구들을 지원할수 있는 비차단교환기 WaveSwitch9200을 생산하였다.

6. 4. 3. SAN/LAN구성을 위한 Myrinet

Myrinet는 Myricom, Inc에 위하여 제안된 Gbps파케트교환망이다.

Myricom의 목표는 콤퓨터들의 클라스터들을 구성하기 위한 체계호상접속상품(제품)을 만드는것이다. 아래에서 Myrinet기술과 교환기들, 련결들, 대면부들, 쏘프트웨어지원, 제안된 응용들을 개괄한다.

Myrinet는 다중콤퓨터와 캘리포니아기술대학에서 개발된 VLSI기술, 북부캘리포니아대학에서 개발된 ATOMIC/LAN기술에 기초하고 있다.

이것은 탁상호스트들과 봉사기 farm들의 in-cabinet SAN클라스터들과 LAN에 기초한 클라스터들을 구성하기 위하여 설계되였다. Myrinet는 임의의 위상을 말을수 있으며 교환기들로 된 그물이나 임의의 정규적인 위상으로 제한할 필요가 없다.

Myrinet는 가변길이파케트형식, 매 련결우에서의 흐름조종과 오유처리, 파케트들의 경로를 정하기 위한 cut-through크로스바교환기들의 사용, 전용프로그람화가능한 호스트대면부들에 의하여 자료련결수준에서 정의된다. Myrinet SAN은 감소된 물리적크기와 부

분들로 하여 Myrinet LAN보다 가격이 낮다.

물리수준에서 Myrinet는 1.28+1.28Gbps의 최대속도를 가지는 완전중복SAN련결들을 3개까지 사용한다. LAN의 련결로서 사용하면 전기케블에서는 25m까지, ribbon빛섬유에서는 500m까지 달성할수 있다.

Myrinet교환기

차단식 cut-through(wormhole)파케트경로정하기는 Gray T3D와 Intel Paragon에 쓰인것과 비슷하게 Myrinet교환기들에서 사용된다.

임의의 망위상에서 다중포구교환기들은 다른 교환기들이나 단일포구호스트대면부들과 련결들에 의하여 접속된다.

그밖에 매 교환기는 그림 6-3에서 보여 준것과 류사하게 흐름조종과 입력완충기들을 가진 판흐름식크로스바이다.

파케트는 머리부가 수신되고 해득되자마자 선택되여 나가는 통로에로 전진한다. 교착허용경로정하기방법에 따라 여러 파케트들이 동시에 Myrinet를 통과할수 있다.

현재의 8개 포구 Myrinet교환기는 10.24Gbps의 2등분대역너비를 가지며 경로형성에 최대로 300ns의 지연이 있고 전원소비는 6~11w이다.

Myrinet파케트의 유효자료는 가변길이를 가진다. 이것은 임의의 형태의(IP와 같은) 파케트를 적응층이 없이 나를수 있게 한다.

Myrinet호스트대면부

호스트대면부는 Myrinet대면부와 파케트대면부, DMA기계, 고속정적RAM을 가지는 LANai소편이라고 부르는 32bit전용VLSI처리기근방에 설치된다.

SRAM은 파케트완충에서와 같이 Myrinet조종프로그람(MCP)을 기억하는데 사용된다.

이 극소형구조는 일반모선과 Myrinet련결사이에 유연하며 고속의 대면부를 제공한다. 지금 Myricom은 Sun SPARC워크스테이션들을 위한 Myrinet/Sbus대면부들과 PCI에 기초한 개인용콤퓨터들을 위한 Myrinet/PCI 대면부들을 만들고 있다.

빛섬유대면부들도 만들어 지고 있다.

MCP쏘프트웨어는 OS부가처리를 피하여 대면부의 처리기에서 실행된다.

그러나 장치구동프로그람과 OS는 여전히 호스트에서 실행된다.

Myricom은 표준 TCP/IP 및 UDP/IP대면부들과 흐름선식(Streamlined)Myrinet API를 제공한다.

실례 6.8. Myrinet련결된 LAN/클라스터구성

그림 6-26에서 탁상워크스테이션들과 PC들, in-cabinet다중콤퓨터클라스터, 단일기판다중처리기클라스터를 접속하기 위한 Myrinet LAN의 구성을 보여 준다. Myrinet LAN은 4개의 Myrinet교환기로 구성된다.

다중콤퓨터 cabinet안에서 2개의 교환기는 하나의 SAN을 형성한다.

망의 RAM과 디스크배렬들은 Myrinet에 소속된다.

이와 같이 Myrinet는 클라스터콤퓨터응용들을 지원할수 있는 큰 잠재력을 가지고 있다. 그러나 모선의존호스트대면부들은 아직도 많은 종류의 호스트들을 Myrinet에 련결하는데서 제한을 가진다.

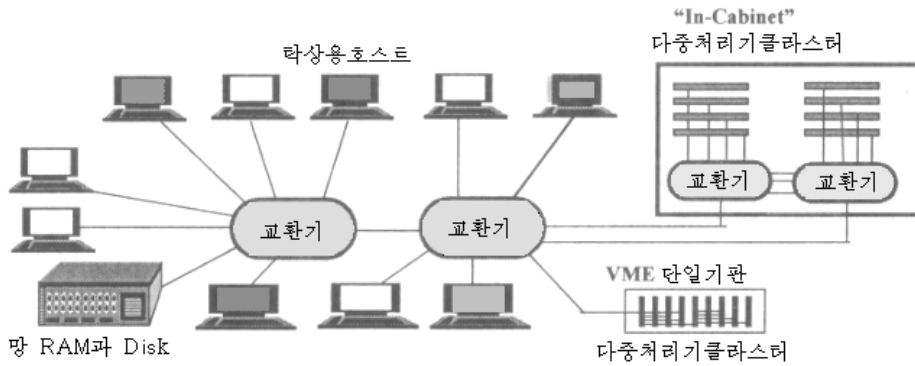



그림 6-26. 4개의 8개 포구교환기로 된 Myrinet 클라스터

25m이하의 동선케블거리는 빛섬유련결이 리용가능하게 될 때까지 SAN과 같이 Myrinet의 리용을 제한한다.

6. 4. 4. HiPPI와 Super HiPPI

HiPPI기술은 서로 다른 종류의 콤퓨터들과 그것의 주변장치들의 망화에 널리 리용되였다. HiPPI는 초고속콤퓨터들만을 위한것이 아니다. 아래에서 HiPPI의 현 상태와 앞으로의 전망에 대하여 평가한다.

HiPPI기술

고성능병렬대면부(High-Performance Parallel Interface, HiPPI)는 서로 다른 판매자들로부터 구입하는 모든 대형콤퓨터들과 초고속콤퓨터들의 대면부를 통일시킬 목적으로 1987년에 Los Alamos National Laboratory이 공개한 표준이다. HiPPI는 대형콤퓨터와 초고속콤퓨터공업에 짧은 거리의 체계 대 체계, 체계 대 주변장치접속을 위한 고속 I/O로서 도입되였다.

1993년에 ANSIX3T9.3위원회는 HiPPI표준을 승인하였는데 이것은 물리 및 자료련결층을 포함한다.

그외의 모든것은 사용자에게 달려 있다.

HiPPI는 800Mbps~1.6Gbps의 속도로 자료를 전송하기 위한 간단한 점대점대면부이다. HiPPI망제품들이 처음에 소개되였을 때 마디당 30000$에서 대중화하기에는 가격이 너무 높았다. 기술이 발전하고 가격이 마디당 4000$이하로 낮아 졌을 때 외부의 초고속콤퓨터사용자들로부터 주의가 돌려 지기 시작하였다.

더우기 ATM과 빛섬유통로, Sonet와의 HiPPI호상운영성이 실현되였다. 이와 같이 고속망화에서 HiPPI의 역할은 아주 중요하며 필요하다.

대면부와 케블요구

기본적인 대면부는 50bit너비인데 32bit는 자료이고 18bit는 조종신호이다. 40ns마다 나온 32bit단어는 전체적으로 800Mbps의 속도를 보장한다.

물리명세서는 25m까지의 거리에서 차폐쌍꼬임선 50쌍을 사용하도록 되여 있다. 이렇게 작은 거리는 작은 망들에서는 충분하지만 기본흐름 LAN들에는 거의나 충분치 않다. 다중방식빛섬유케블은 HiPPI표준에서 빛섬유확장기(fiber extender)를 사용하지 않고 300m에서 동작한다. 더 먼 거리에 대하여 빛섬유확장기들은 다중방식에서는 10km까지, 한파모습에서는 20km까지 지원할수 있다. 간단한 HiPPI통로들에서 두 통로 통신을 보장하기 위해서는 2개의 케블이 필요하다. 1.6Gbps를 달성하기 위하여 4개의 케블들이 완전중복으로 동작할수 있다.

HiPPI는 50쌍의 UTP를 사용하기때문에 공통기업망하부구조(common enterprise network infrastructure)들에서 찾아 볼수 있다. 이것은 ATM과 빛섬유통로와 같은 다른 고속망들과 비교되는 명백한 부족점으로 된다.

실례 6.9. 전형적인 인터네트망구성

그림 6-27에서 설명하는바와 같이 쌍방향HiPPI크로스바교환기들은 여러가지 대형콤퓨터들과 봉사기들, 초고속콤퓨터들사이의 전용접속을 실현하기 위하여 개발되였다.

HiPPI-Serial은 빛섬유로 10km까지 접속할수 있다. 두개의 HiPPI교환기들이 HiPPI기간을 형성하기 위하여 련결된다. 교환기들이 직렬(serial)대면부들로 장비하지 않으면 긴 련결을 위하여 HiPPI확장기가 필요하다.

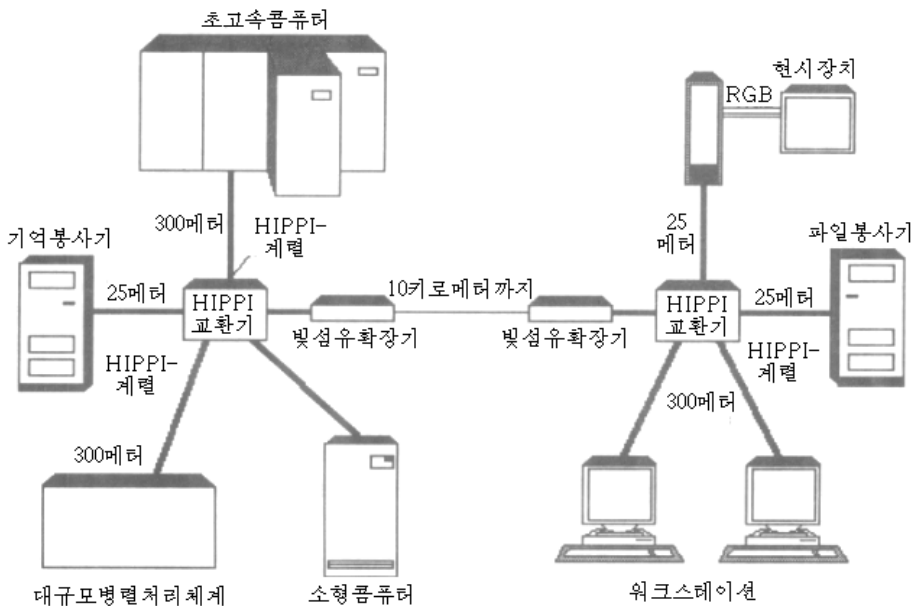



그림 6-27. LAN 호상접속의 기본기간으로서 HiPPI 통로들과 HiPPI 교환기들의 사용

TCP/IP망들과의 인터네트망화는 RFC1347로 주소화된다.

대부분의 HiPPI교환기판매자들은 TCP/IP구동프로그람들도 제공한다.

그러나 HiPPI의 접속지향속성은 방송전송실현의 절충을 요구한다.

교환기들에서 ATM LANE와 같은 주소봉사기를 사용하는가 아니면 본래대로 방송요

구를 처리하는가는 해결해야 할 문제로 남아 있다.

HiPPI의 통로와 교환기

HiPPI통로들은 고속 I/O 또는 주변통로로서 소개되였다.

그러나 집단내방송은 지원하지 않는다.

상업기계들에서 HiPPI통로들과 HiPPI교환기들은 SGI PowerChallenge봉사기클라스터와 IBM390대형콤퓨터들, Gray Y/MP, C-90, T3D/T3E체계들에서 사용된다.

빛섬유통신과 류사하게 HiPPI는 낮은 지연과 동적 및 호상작용적인 사용에는 적당치 않다.

HiPPI통로들 또는 HiPPI교환기들의 일부 동작특징들을 아래에 개괄한다.

일부는 존재하는것이고 일부는 앞으로의 목표이다.

- **초고속자료전송** 지금 HiPPI는 800Mbps 또는 1.6Gbps의 속도로, 단순 및 완전중복으로 구성될수 있다.
- **매우 단순한 신호렬** 기본적으로 Mbps접속은 3가지 통보문 즉 요청(원천이 접속을 요구한다.), 접속(목적지는 접속이 실현되였음을 알린다.), 준비(목적지는 파케트흐름을 접수하기 위한 준비가 되여 있음을 알린다.)로 구성될수 있다.
- **규약독립성** HiPPI통로들은 이른바 생(raw) HiPPI (아무런 웃층규약도 사용하지 않는 프레임화규약으로 형식화된 자료)와 TCP/IP데타그람(datagramme)들, IP I-3(지능주변장치대면부)프레임화된 자료를 처리할수 있다. IP I-3은 RAID(redundant array of inexpensive disks)장치와 같은 주변장치들을 콤퓨터에 접속하기 위하여 사용되는 규약이다. 따라서 HiPPI는 이써네트와 FDDI, 고속자료기억 및 검색을 가지는 인터네트망화와 동등하게 적응할수 있다.
- **물리층흐름조종** HiPPI는 서로 다른 속도로 동작하는 장치들사이의 믿음직하고 효률적인 신용에 기초한 체계를 제공한다. 사실상 원천은 준비(ready)신호들의 행로를 보존하고 목적지가 그것들을 처리할수 있을 때만 자료를 보낸다.
- **련결지향회선교환** 비차단식회선교환기들은 여러개의 회화가 동시에 진행되도록 한다. 그리하여 HiPPI교환기들의 집합대역너비는 800Mbps 또는 1.6Gbps의 배수이고 포구당 대역너비는 포구수의 배수와 같다.
- **동선과 빛섬유의 조화** HiPPI는 짧은 거리에 50쌍의 STP들을 사용한다. 이것은 구내 또는 지역에서 한파모습과 다중방식으로 동작한다. Sonet는 먼거리통신에 사용된다.

HiPPI규약

HiPPI규약은 표준들의 모임에 의해 여러층에서 정의된다. HiPPI-PH표준은 물리층에서 기구적 및 전기적, 신호적으로 단일한 점대점련결을 정의한다.

HiPPI는 단순한 규약이므로 완전중복련결은 반대방향의 다른 HiPPI접속에 의해 달성된다. HiPPI-FP(Framing Protocol)는 사용자정보의 매 파케트들의 형식과 내용(머리부포

함)을 서술한다.

HiPPI−SC는 많은 콤퓨터들이 자료를 공유할수 있게 하는 하나의 동작방법으로 개발되였다.

이것은 여러개의 점대점접속들이 동시에 일어 나도록 할수 있는 교환기구를 설치하도록 요구한다.

HiPPI−SC는 임의의 교환하드웨어는 서술하지 않고 사용되는 장치에 대한 기능적기구들만을 제공한다.

HiPPI를 다른 규약으로 넘기기 위하여 다음과 같이 련결교갑화를 위한 3가지 표준들이 개발되였는데 표 6−5에 개괄하였다.

- HiPPI−LE(link encapsulation)는 IEEE802.LLC의 머리부(header)들의 D1_Area 및 D2_Area의 시작에로의 넘기기를 제공한다.
- HiPPI−FC(Fiber Channel)는 빛섬유통로제품들을 HiPPI−FP표준으로 넘긴다.
- HiPPI−IPI(disk & tape commands)는 IPI−X표준명령모임들을 HiPPI−FP머리부들로 넘긴다. 이 표준은 IPI표준들에 통합된다.

SuperHiPPI

최근 SuperHiPPI기술은 더우더 높은 지연을 가지고 HiPPI보다 8배 빠른 속도를 보장하기 위하여 개발되였다.

이 SuperHiPPI기술은 일반적인 HiPPI사용자들의 일부 소원을 새롭게 하였다. SuperHiPPI기술은 완전히 다른 실현을 제공하며 따라서 HiPPI통로들과의 뒤쪽 호환성이 부족하다. 이것은 HiPPI통로들이나 HiPPI교환기들에 대한 갱신의 새로운 시작을 암시한다.

표 6−5 HiPPI규약묶음조

<table>
<tr><td>HiPPI−LE련결교갑화
(IEEE802.2)</td><td>HiPPI−FC
(빛섬유통로규약들)</td><td>HiPPI−IPI
(IPI−3명령모임들)</td></tr>
<tr><td colspan="3">HiPPI−FP프레임화규약(파케트형식, 파케트머리부들)</td></tr>
<tr><td colspan="2">HiPPI−PH, 기구, 전기, 신호
(물리층)</td><td>HiPPI−SC,교환기조종
(물리장치를 위한것)</td></tr>
<tr><td colspan="2">HiPPI−Serial확장, 빛섬유식 HiPPI−PH 10km
확장기(ANSI표준은 아니다.)</td><td></td></tr>
</table>

SGI나 Los Alamos National Laboratory는 다 25.6GB/s보다 더 높은 HiPPI교환기들을 만드는 HiPPI기술을 연구하였다. Washington D.C에서 열린 ACM초고속콤퓨터94(supercomputing 94)에서는 18마일의 다중방식케블로 구성된 빛섬유(all−fiber) HiPPI기간과 16명의 충돌자들이 90Gbps이상의 집합대역너비를 구성할수 있다는것을 증명하였다.

6. 5. ATM 교환기와 망

비동기전송방식(ATM)은 ATM Fornm(1991년에 창립)과 ITU표준들에서 정의된다.

ATM기술과 세포형식, ATM충돌을 ATM교환기들과 빛섬유케블을 사용하는 인터네트화망의 실례와 함께 아래에 서술한다.

ATM망은 Telecom협회들에 의해 시작되였으며 인터네트기간에서는 MAN이나 WAN으로 출현하였다. 지금 주요콤퓨터협회들은 인터네트나 LAN지원을 위한 자체의 ATM망화지원을 가지고 있다.

6. 5. 1. ATM기술

ATM은 통신흐름을 짧은 고정길이 53B세포들로 토막화하는 매체독립정보전송규약이다.

그 바탕기술은 6.1.3에서 론의한 세포교환에 기초하고 있다. ATM의 목표는 실시간(지연에 예민한) 및 폭발자료(지연에 예민하지 않은)전송능력을 하나의 단일망기술에 통합시키는것이다.

ATM은 광대역망응용들에서 더 큰 경제성과 유용성에 필요한 동적대역너비배정과 고속교환을 제공한다.

ATM은 말단국들이 공통의 매체를 공유하지 않는 교환구조를 채택하며 따라서 LAN과 WAN의 요구들을 다 지원한다.

ATM은 짧은 세포들을 리용하기때문에 높은 속도를 가지는 규소섬유교환기를 만들수 있게 한다.

ATM의 세포형식

그림 6-28에서 서술한바와 같이 긴 파케트는 ATM망으로 전송되기전에 세포들로 쪼개져야 한다.

수신단에서 세포들은 본래의 파케트형태를 재생하도록 재조립된다. 첫 5B는 경로정하기를 위한 세포머리부이다. 유효자료는 최대 48B의 자료정보를 포함한다. 작은 세포들과 가상경로폭, 가상통로들의 사용은 ATM세포교환기설계를 합리적이게 한다.

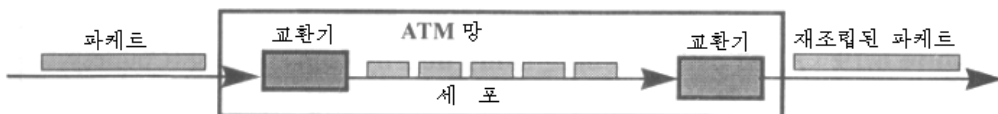
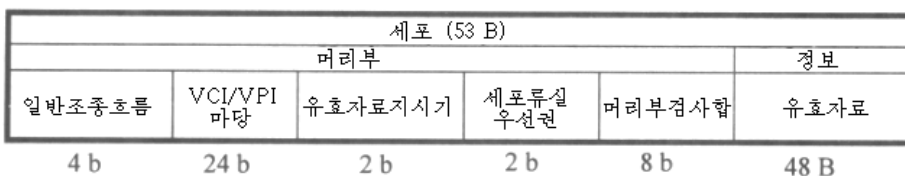



그림 6-28. ATM세포교환의 개념과 세포형식

ATM세포들은 원천경로정하기방법(source routing scheme) 또는 hop-by-hop방법을 사용하여 로정을 정할수 있다. 원천경로정하기를 위하여 전체 경로정보는 세포의 머리부에 포함되여야 한다.

그러므로 원천경로정하기에서 경로기는 길이의 제한을 받는다. hop경로정하기에서는 머리부가 hop ID만을 요구하며 결과 경로에 대한 보다 융통성 있는 선택이 가능하다. 대부분의 ATM교환기들은 우에서 보여 주는 세포형식을 사용하는 hop-by-hop경로정하기방법을 선택하고 있다.

세포교환기설계

파케트교환망의 전통적인 사용을 떠나 ATM은 세포교환망을 구성하기 위한 새로운 표준을 제공한다.

세포교환망은 한개 망우에서 음성으로부터 화상, 동화상, 자료에 이르기까지 모든 매체형태를 고속전송하도록 설계된다. ATM은 표준통신규약들을 사용하는 현재의 LAN들과 MAN들, WAN들과 통합되여야 한다. ATM세포망들은 광대역 및 다매체정보를 전송하는데 사용되였다. 특히 ATM는 집단내방송조작들을 효과적으로 수행할수 있어야 한다.

그림 6-29에서 hop-경로정하기방법에 기초한 세포교환기설계의 개념을 보여 준다.

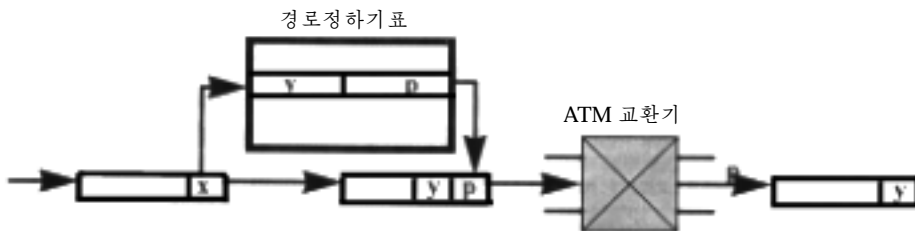



그림 6-29. hop-by-hop경로정하기를 사용하는 ATM세포교환

ATM교환기에 들어 가는 매 련결에서 도착하는 세포들의 가상경로식별자 UPI(virtual path indentifier)와 가상통로식별자 VCI(virtual channel indentifier)의 값들은 세포가 지나가야 할 경로(path)와 통로를 유일하게 결정한다.

새로운 가상식별자들은 나가는 련결의 세포머리부에 배치된다. 경로정하기표들은 가상경로구성절차에 의해 초기화된다.

ATM교환기에 들어 간후 세포의 VPI마당(x)은 경로정하기표로부터 입력자료를 선택하는데 사용된다.

새로운 VPI값(y)은 다음교환기로 가는 세포에 배치된다. 같은 가상경로우에서 두개 호스트들은 세포의 VCI마당들을 리용하여 가상통로흐름의 무리들을 구분함으로써 많은 응용흐름들을 다중화할수 있다.

ATM의 속도

지금 ATM망들은 25Mbps로부터 51,155,622Mbps까지의 다양한 속도를 지원한다. 속도가 낮을수록 ATM교환기들과 사용된 련결들의 가격은 낮아 진다. ATM의 속도는 앞으

로 더 높아 질것이다. 포구당 622Mbps를 가지는 FORE체계 ASX1000ATM은 10Gbps의 집합대역너비를 실현할수 있다.

6. 5. 2. ATM망대면부

ATM세포들은 2개 수준의 계층(two-level hierarchy) 즉 가상경로와 가상회선에서 교환된다. 물리적련결안에 여러개의 가상경로들이 상주할수 있다.

매개 가상경로에는 1개이상의 가상회선이 존재할수 있다. 가상회선들은 호스트들에 접속하는 ATM교환기들에 의해 인식되며 교환기들사이에는 가상경로가 사용된다. 때문에 그림 6-30에서 서술하는바와 같이 사용자망대면부(User-Network Interface, UNI)와 망 대 망대면부(Network-Network Interface, NNI)가 정의된다.

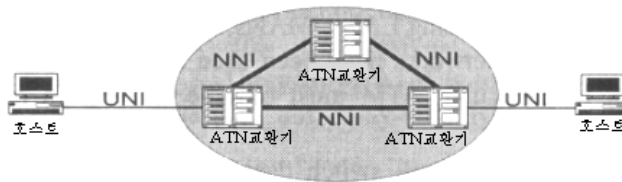



그림 6-30. ATM망의 사용자망대면부(UNI)와 망 대 망대면부(NNI)

2개 수준의 교환계층은 하드웨어고장으로 일부 교환기들이 가상경로를 다시 정할 때 우월성을 가진다. 이때 모든 관련가상회선들은 경로기들을 개별적으로 재구성하지 않고 자동적으로 다시 경로가 정해 진다.

5B의 ATM층머리부는 그림 6-31에서 보여 주는바와 같이 UNI와 NNI에서 서로 다른 형식을 가진다. 주요차이는 UNI에는 일반흐름조종(Generic Flow Control,GFC)마당이 존재하고 NNI에는 더 큰 VPI마당이 존재한다는데 있다.

GFC마당은 종단국을 확인하며 교환기는 가상회선이 충돌상태에 있으면 한개 세포를 다른데로 먼저 통과시킨다.

UNI에서 ATM층의 머리부

GFC (4-bit)	VPI (8-bit)	VCI (16-bit)	PTI (3-bit)	CLP (1-bit)	HEC (8-bit)

NNI에서 ATM층의 머리부

VPI (12-bit)	VCI (16-bit)	PTI (3-bit)	CLP (1-bit)	HEC (8-bit)

GFC: 일반흐름조종
VPI: 가상경로식별자
VCI: 가상통로식별자
PTI: 유효부하형태식별자
CLP: 세포류실우선권
HEC: 머리부오유검사

그림 6-31. ATM세포들에서 UNI와 NNI의 머리부형식

기정설정은 모두 0이다.

세포가 첫번째로 도착하는 교환기에 의하여 기정설정이 다시 써지므로 끌 대 끌은 무의미하게 된다.

NNI세포들의 더 큰 VPI마당에 의하여 교환기들은 경로정하기에서 많은 경로들을 리용할수 있다.

ATM Forum은 서로 다른 판매자들의 교환기들을 사용할수 있게 하는 사설망 대 망 대면부표준도 개발하였다.

일부 ATM교환기와 망제품들을 속도와 포구수, UNI 또는 NNI의 지원에 따라 보면 표 6-6과 같다.

FORE체계의 ATM교환기와 련결들을 리용하여 LAN기간을 구성하는 방법은 후에 보여 준다.

6. 5. 3. ATM구조의 4개 층들

여기서는 계층화된 ATM망과 OSI(Open Systems Interconnection)표준과의 대응, ATM 세포정보처리를 서술한다.

표 6-6 기가비트ATM교환기와 망들의 실례

제품	집합대역너비	포구의 수	UNI와 NNI지원
Bay Networks Centillion 100 ATMspeed/155	1.2Gbps		
Cisco LightStream 190	5Gbps, 공유기억기차단	64	V3.0과 V3.1
DEC GIGAswitch/ATM	3.6Gbps	22	
FORE ASX-1000	10Gbps, 비차단	96	완전 V3.0
2230Nway 600 과 650	5Gbps, 비차단	8	V3.0과 V3.1

계층화된 ATM모형

개념적으로 ATM망은 그림 6-32과 같이 4개의 층으로 된 3차원모형으로 서술된다. 물리층은 2개의 ATM부분층(TC와 PMD)으로 세분화된다. ATM층은 끌 대 끌 가상회선교환을 지원하는 망층에 대응된다. ATM층은 접속지향이므로 임의의 응답기구도 제공하지 않는다.

ATM적응층은 CS와 SAR부분층으로 갈라 진다. 웃쪽 층들은 사용자판(plane)과 조종판(plane)에 가깝다. 세번째 차원은 층과 판관리구조들을 보여 준다. 리론적으로 ATM세포의 재전송은 고도로 믿음직한 빛섬유망의 리용에 기초하고 있지 않았다. 오유처리의 과제는 웃쪽의 층들에 남겨 둔다. 더우기 음성과 영상과 같은 실시간응용들에서는 전체 화상을 재전송하기보다 틀린 세포를 허용하는것이 더 좋다.

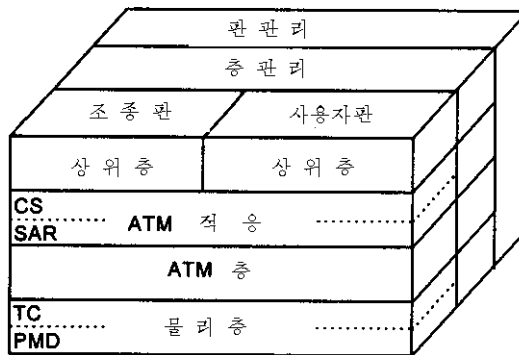



그림 6-32. ATM망의 구조

대응하는 OSI의 층들

표 6-7은 ATM부분층들의 기능과 ISO가 제안한 OSI망표준의 7개 층과의 대응을 정의한다. 흥미 있는 독자들은 ATM과 ISO표준을 상세히 참고하면 된다.

표 6-7 ATM층의 기능과 OSI층과의 대응

OSI층	ATM층	ATM부분층	기능
3/4	AAL(ATM adaptation Layer)	CS	표준대면부제공
		SAR	토막화와 재조립
2/3	ATM		흐름조종, 세포머리부의 생성/추출, 가상회선/경로관리, 세포다중화/비다중화
2	물리층	TC	세포속도분리, 머리부검사합생성과 비교, 세포의 생성, 외피로 세포를 씌우기(packing)/풀기
1		PMD	bit시간화, 물리망접근

세포의 형성

나가는 세포에 대하여 ATM층은 더 높은 층 AAL로부터 48B의 유효자료를 받아서 5B의 머리부를 더한 다음 물리층으로 통과시킨다. 들어 오는 통신흐름에 대해서는 이와 꼭 반대이다. 이 세포형성과정을 그림 6-33에서 설명한다.

ATM층은 같은 가상회선을 따라 도착한 세포들의 정확한 순서를 담보한다. 혼잡으로 망이 세포들을 잃어 버려도 단일한 가상회선으로 보낸 세포들의 재배렬은 허용되지 않는다. 서로 다른 가상회선으로 보낸 세포들에 대해서는. 이러한 담보가 없다.

중간의 교환기가 일시적인 과부하로 한개의 세포를 잃어 버릴 때 전체 파케트가 파괴되여 긴 지연이 생긴다. 믿음직한 세포넘기기는 ATM층의 제일 웃층에서 하여야 한다. 더구나 최량적인 세포크기는 서로 다른 사용자집단들사이에서 아직까지 토론중에 있다.

ATM을 지난 날에는 주로 전화망기술에 사용하였다. 그러나 더 작고 국부화된 망들에서 ATM기술을 리용하는 경향도 있다.

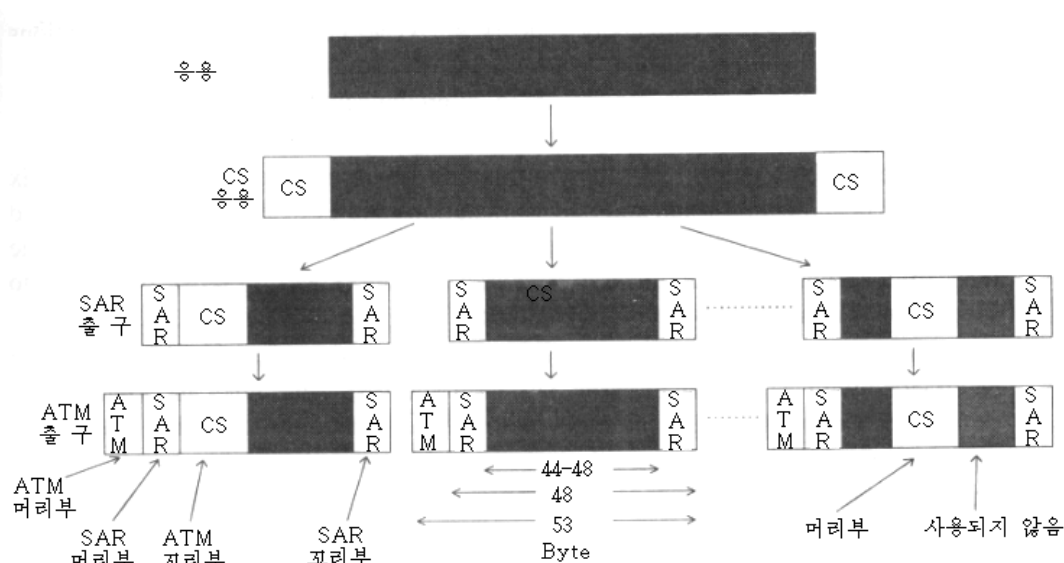



그림 6-33. 나가는 세포에서 응용유효자료의 련속적인 ATM 층통과

6. 5. 4. ATM의 망간접속성

여러가지 ATM교환기들과 련결들, 적응기들이 서로 다른 망환경에 사용될수 있다. 실례로 작업실내의 PC들과 워크스테이션들, 봉사기들을 접속하기 위한 전용ATM교환기들을 구성할수 있다.

ATM교환기는 LAN기간을 구성하거나 유산의 LAN들(이써네트나 통표고리, FDDI고리)에 접근하며 영상을 워크스테이션의 클라스터나 인터네트접근을 가지는 인트라네트에 배포하기 위하여 사용할수 있다.

아래에서 전용인터네트봉사응용들을 제공하는데서 분산다매체처리를 위한 ATM에 기초한 인트라네트의 설계를 보여 준다.

실례 6.10. Pearl클라스터 : HKU의 ATM에 기초한 다중콤퓨터클라스터

홍콩대학(Hong Kong University, HKU)에 ATM에 기초한 워크스테이션들과 봉사기들의 클라스터가 구성되였다.

그림 6-34에서 ATM망의 구성을 보여 준다.

클라스터는 5Gbps의 집합대역너비를 가지는 호상접속된 3개의 ATM교환기주변에 구성된다.

155Mbps다중방식섬유를 사용하는 망케블이 6개의 SMP봉사기들과 40개이상의 Sun 및 SGI워크스테이션들, PC들을 호상접속하는데 사용되며 이 모두는 같은 건물안에 위치한다. 클라스터의 모든 호스트들은 HARNET(Hong Kong Academic Research Network)와 WWW자원들에 접근하기 위한 인터네트기간을 거쳐 밖의 자원에 접근할수 있다.

ATM련결과 비ATM련결이 다 사용된다. 모든 ATM비호환호스트들은 ATM적응기기판들과 련결되여야 한다. 서로 다른 속도의 외부련결들이 Pearl클라스터에로 또는 Pearl

클라스터로부터 접속된다.

그러나 내면적으로 클라스터내에서는 현재의 이써네트와 FDDI접속외에 155Mbps의 빛섬유케블만이 사용된다. 대부분의 호스트들은 ATM기간교환기(ASX−1000, ASX−200BX)들에 접속된다.

이써네트에 련결된 일부 호스트들은 이써네트집선기(Power Hub 7000)를 통하여 간접적으로 접속된다.

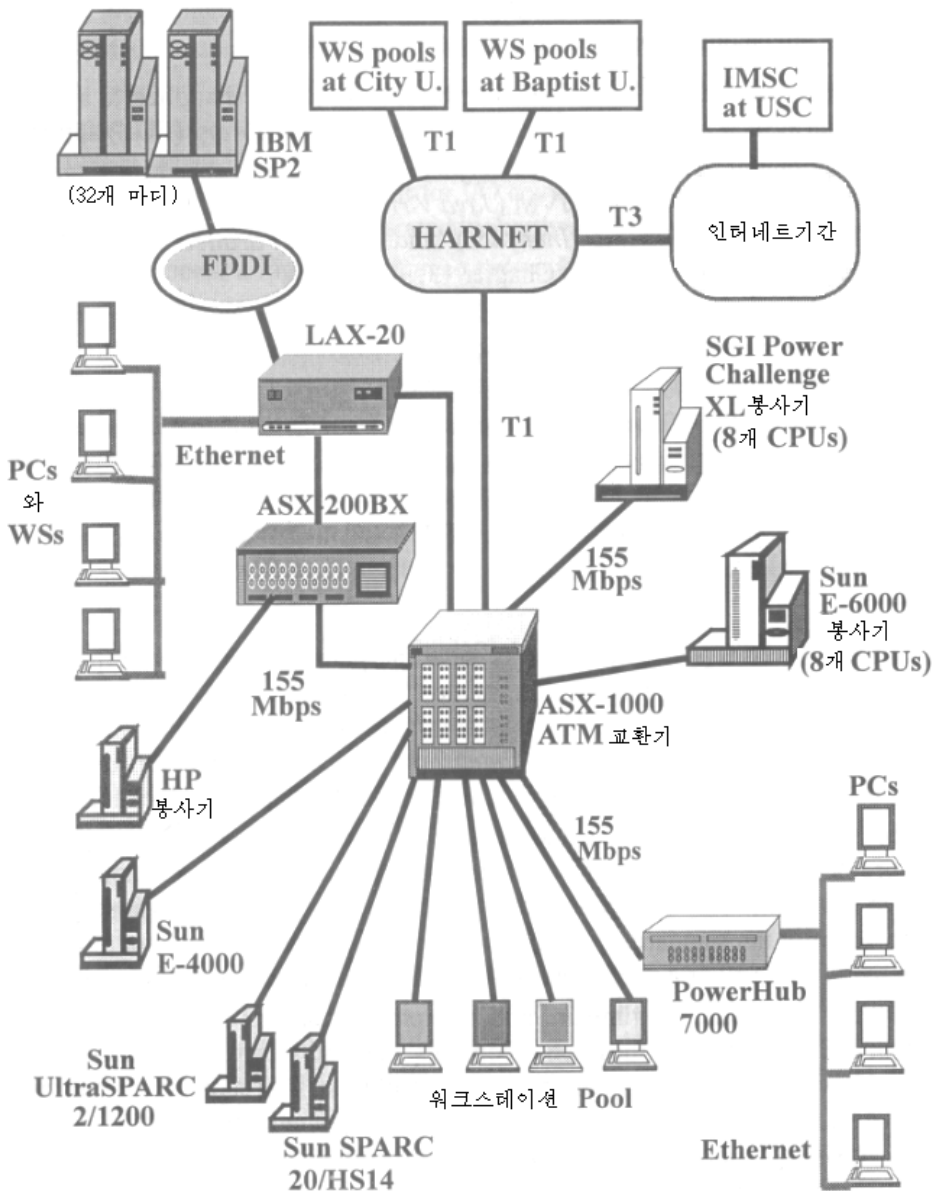



그림 6−34. Pearl 클라스터 : 분산다매체 및 메타콤퓨터응용전용인
홍콩대학의 봉사기들/워크스테이션들의 ATM 클라스터

LAX-20LAN접근교환기는 이써네트와 FDDI고리를 접속한다. ATM교환기, 155Mbps빛섬유련결들, 적응기기판들은 HKU클라스터를 구성하는데 사용되는 기본적인 구성요소들이다.

Pearl클라스터의 응용

Pearl클라스터의 능력에 기초한 연구과제에는 특수한 클라스터의 구조적개발, 분산다매체처리, 메타콤퓨터리용, 금융수자서고 등이 포함된다.

(1) Pearl클라스터의 일부 부하공유능력(Load Sharing Facility, LSF)은 금융수자서고(Financial Digital Library,FDL)응용들에서 단일체계영상(Single System Image, SSI)의 조작들을 실현하는데 응용된다.
(2) 13장에서 론의하는 통보문넘기기대면부(MPI)와 병렬가상기계(Parallel Virtual Machine,PVM)표준들외에 능동통보문(active message)들과 쟈바가상기계(Java Virtual Machine, JVM)대면부들을 사용하는 고속통보문넘기기를 위한 MPI자바대면부(MJI)의 개발.
(3) 클라스터는 Web자원으로부터 영어와 중어문서들 또는 다매체파일에 대한 고속정보검색을 위한 두 나라말을 쓰는 WWW색인과 탐색엔진으로 설계된다.
(4) 클라스터성능검사를 위한 TPC(Transaction Processing Council)성능평가기준들의 확장과 새로운 분산다매체성능평가기준의 생성.
이 TCP 및 DMS성능평가기준들은 상업적응용들에서 클라스터화된 초고속봉사기의 사용능력을 평가하도록 설계된다.

특정령역과 부족점

ATM망들은 수자전화와 의료형태, 업무처리, 요청텔레비죤과 분산다매체처리에 광범히 사용할수 있는 능력을 가지고 있다.

원격통신공업은 많은 광대역ISDN(Integrated Services Digital Network)봉사들을 실현하는데 이미 ATM을 선택하였다.

ATM의 미래는 LAN 및 WAN기술과 통합하는 능력에 달려 있다. 부정적인 측면에서 ATM망들은 공동으로 수집한 세포관리와 망규약들이 부족하다.

전화협회와 콤퓨터공업은 지금 서로 다른 표준들을 적용하고 있다.

ATM협회는 표준들을 단일화하는것을 목적으로 하는 국제적인 조직이다.

실례로 ATM망은 자료련결층규약을 제공하므로 세포나 파케트의 믿음직한 넘기기를 담보할수 있다.

콤퓨터와 다매체, 원격통신공업이 같은 ATM표준들의 모임에 동의하지 않으면 이 부족점들은 ATM기술의 리득을 릉가할수 있다. 지어 ATM은 광대역 또는 다매체지향의 MAN들과 WAN들, 그리고 지역들의 인터네트화에서도 자기의 유용성을 증명하였다. 기가비트이써네트와 다른 낮은 가격의 LAN기술들보다 높은 ATM의 가격은 LAN들에서 ATM의 광범한 사용을 제한한다.

6. 6. 확대가능한 일관성대면부

우에서 론의한 기가비트 및 ATM망들은 한가지 공통점을 가진다. 즉 이것들은 모두 마디의 I/O모선에 접속되며(그림 6-2를 참고) 통신은 콤퓨터마디들사이에 통보문을 보냄으로써 진행된다. Gustavson과 Li[289]는 이러한 형태의 I/O통신이 모선에 기초한 SMP의 공유기억기통신보다 렬등하다는것을 지적하였다.

공유기억통신에서 통신의 하나의 처리기는 store지령을 수행하고 다른 처리기는 load지령을 수행하는것에 지나지 않으므로 낮은 지연을 가진다. I/O통신에서는 수백수천의 지령들이 수행되여야 한다. 더우기 I/O통신이 모선에 기초한 SMP의 하드웨어에 의하여 생성되는 캐쉬일관성정보의 우월성을 가지기는 힘들다.

이러한 원인들로 하여 고성능망은 기억기모선(국부모선)에 대면부로 접속되여야 한다. 전통적인 망들은 모선에 비하여 2가지 주요우점을 가진다.

- 일반적으로 망들은 표준이고 상대적으로 견고하지만 기억기모선은 그렇지 않다. 이써네트는 10년이상 표준화되였으며 오늘 더 널리 사용되고 있다. 그러나 기억모선은 새로운 처리기가 나오는 18달마다 변한다.
- 전통적인 망들은 수십, 수백메터로 공간확대가능하지만 기억모선은 겨우 1m까지 확장할수 있다. 짧은 길이는 SMP체계가 지원할수 있는 처리기의 수를 제한한다. 모선의 우월성을 유지하면서 전통적인 망의 공간확대가능성을 가지는 표준적인 호상접속구조를 설계하는것이 가능하겠는가?

그 대답은 IEEE/ANSI의 표준 1596-1992이다.

즉 확대가능일관성대면부(Scalable Coherence Interface,SCI)는 틀에 박힌 뒤판(backplane)모선을 완전 2중점대점호상접속구조로 확장함으로써 분산공유기억기의 일관성캐쉬형태를 제공한다. SCI는 풍부한 규약이다(기본명세서는 248페지로서 길다.).

아래에서는 SCI가 모선 즉 기본적인 자료전송규약으로부터 어떻게 발전되며 캐쉬밀착을 어떻게 강화하는가를 론의한다. SCI체계실현의 실례는 8장에서 론의한다.

6. 6. 1. SCI호상접속

SCI는 낮은 지연(1㎲이하)과 높은 대역너비(8GB/s까지)의 점대점호상접속을 제공하도록 설계된다. 완전히 개발되면 SCI는 64k개까지의 마디들을 접속할수 있다. 가장 최근의 SCI표준(IEEE 1596-1996)은 250MB/s로부터 8GB/s까지의 대역너비를 기록하고 있다. SCI련결들은 빛섬유케블은 물론 동선으로도 실현될수 있다. 대면부소편은 CMOS와 BiCMOS, GaAs기술로 실현한다.

SCI의 캐쉬일관성특징은 련결된 SCI모듈들의 목록으로 실현된다. 매 처리기마디는 하나의 SCI모듈에 소속된다. 처리기가 자기의 캐쉬를 갱신하면 그 캐쉬의 상태는 같은 캐쉬행을 공유하는 모든 SCI모듈들을 따라 전파된다. 분산일관성캐쉬들에 대하여 련결된 이 목록은 확대가능하다. 왜냐하면 더 많은 SCI모듈들을 련결된 목록을 따라 삽

입하여 큰 체계를 창조할수 있기때문이다.

호상접속위상

SCI는 마디들과 외부호상접속사이의 대면부를 정의한다. 최초의 목표는 련결당 1GB/s의 대역너비를 가진 16bit련결들을 사용하는것이였다.

결과 뒤쪽 판(backplane)모선들은 단일방향점대점련결들로 교체되였다.

SCI호상접속들은 일반적으로 위상독립이다.

매 SCI마디는 련결된 기억기와 I/O장치들을 가지는 처리기일수 있다. SCI호상접속은 임의의 위상도 가능하다. 매 마디는 SCI고리들 또는 크로스바로부터 접속되거나 SCI 고리들 또는 크로스바에로 접속되는 하나의 입력련결과 하나의 출력련결을 가진다. SCI련결의 대역너비는 련결과 대면부를 실현하기 위하여 선택한 물리적표준에 의존한다.

SCI환경에서 방송과 모선에 기초한 작용의 개념은 포기되였다.

일관성규약들은 요청자에 의해 시작되고 응답자에 의해 완성되는 점대점작용에 기초한다. 고리호상접속은 마디들사이의 가장 단순한 귀환접속들을 제공한다.

모선으로부터 SCI고리로

그림 6-35에서는 모선호상접속으로부터 확대되는 전형적인 SCI고리를 보여 준다. SCI를 도입하게 된 동기를 리해하기 위하여 확대가능한 체계를 구성할 때 모선에서 제기되는 3가지 문제들을 살펴 볼 필요가 있다.

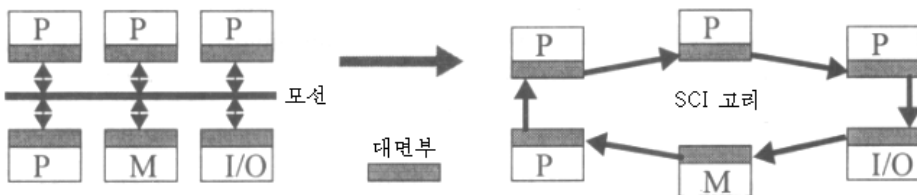



그림 6-35. 수자모선으로부터 SCI 고리의 발전

- **신호화문제** 모선전송선들은 여러가지 처리기와 기억기, I/O장치들을 접속하는 분기들을 가지고 있기때문에 완전하지 않다. 이것은 반사와 잡음 특히 모선구동장치들이 많은 전류를 요구하는것과 관련된다.
- **병목문제** 모선은 한번에 하나의 송신기만이 사용할수 있는 공유매체이다. 분산작용규약들이 조금 도와 주지만 모선중계와 주소화는 항상 매 작용에 대하여 진행되여야 한다.
- **크기문제** 신호화의 어려움으로 하여 고속모선은 짧아야 한다. 짧은 모선은 접속될수 있는 장치들의 수를 제한한다.

SCI는 모선과 관련한 이상의 3가지 문제들을 극복하기 위하여 그림 6-35에서 보여주는바와 같이 다음과 같은 기술들을 채택한다.

- **점대점련결** SCI는 여러가지 처리기와 기억기, I/O장치들을 마디로 간주한다.

그리고 하나의 송신마디로부터 수신마디에로 서로 다른 신호화를 가지는 점대점 련결을 사용한다. 접속분기들이 더는 없으며 중요하게는 반사/소음문제가 크게 완화되여 신호화속도가 크게 늘어 나도록 한다. SCI는 처리기들과 기억기, I/O장치들을 포함하는 더 복잡한 마디들을 배제하지 않는다.

- **한방향고리** 련결들은 련속적으로 그리고 한 방향으로 동작한다. 이것은 구동기 전류를 일정하게 하여 잡음을 더욱 줄이게 한다. 매 마디는 적어도 하나의 입력 련결과 하나의 출력련결을 가져야 하므로 한방향고리는 가장 간단한 위상이다.
- **병행** 한번에 하나의 작용만이 사용할수 있는 모선과 달리 여러 마디들이 SCI고리에로 파케트들을 동시에 주입하고 추출할수 있다.

고리로부터 그물에로

2차원그물과 같은 다른 위상들은 그림 6-36에서 설명하는것과 같이 여러개의 고리들로 구성될수 있다. 대면부모듈들은 고리들로 된 그물에 다리를 놓는다. SCI호상접속의 대역너비와 중계, 주소화기구들은 뒤쪽 판 모선(backplane bus)들을 크게 릉가할것이 예상된다. SCI의 기본적인 우월성은 그것의 확대가능성에 있다.

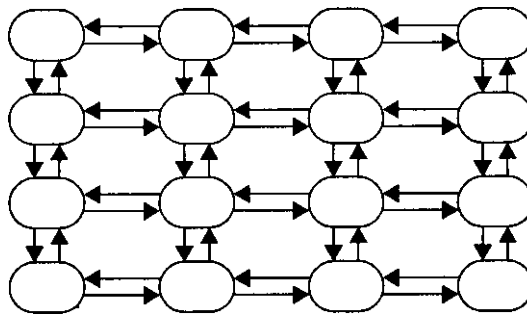



그림 6-36. 여러 SCI 고리로 된 2차원그물

6. 6. 2. 실현문제

SCI표준은 많은 콤퓨터제작자들로부터 지지를 받았다. 주목할만한 실례로서 HP/Gray와 IBM, Data General, Sequent, SGI/Convex들을 들수 있다.

SCI구조의 대부분의 응용들은 8장에서 고찰한다.

이 부분에서 련결형식과 파케트형식, 마디대면부들, SCI박자화와 같은 일부 SCI실현사항들을 고찰한다. 마지막에 SCI실현의 실례를 보여 준다.

련결과 파케트형식

SCI는 파케트에 기초한 분산작용규약들을 사용한다. SCI호상접속에 사용된 SCI련결과 파케트들의 형식을 그림 6-37에 주었다.

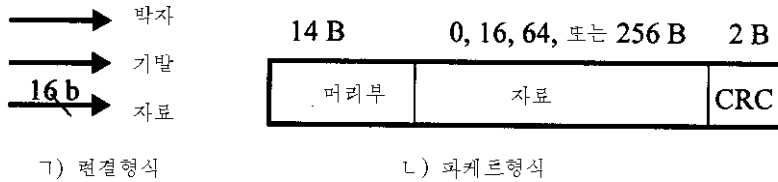



그림 6-37. SCI 련결과 파케트형식들

SCI마디대면부

마디대면부는 그림 6-38에 상세히 제시한다. SCI련결은 1개의 박자비트, 1개의 기발비트, 16개의 자료비트를 가지는 18개 비트폭이다. 그러한 18개 비트를 **기호**라고 부른다. 파케트는 14B의 머리부와 오유처리를 위한 2B의 CRC, 0 또는 16, 64, 256B의 자료를 포함한다.

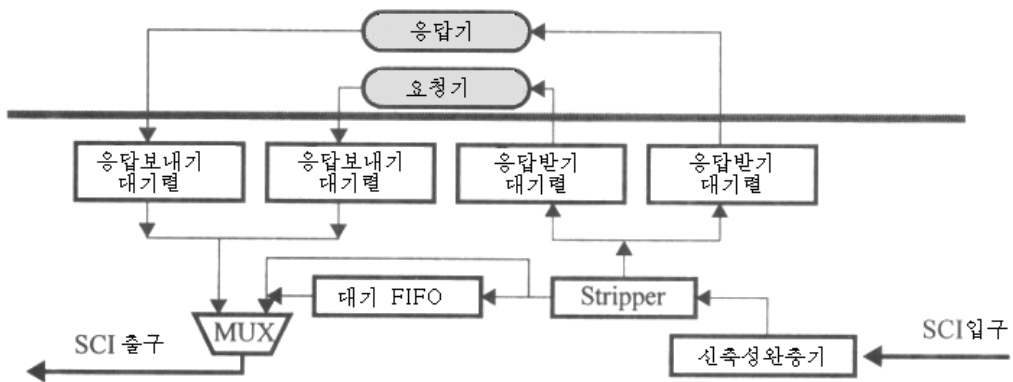



그림 6-38. 단순화된 SCI 마디대면부의 구조

매 SCI박자주기에 한개의 파케트기호(2B)가 전송될수 있다. 박자와 기발비트는 하드웨어가 생성한다. 그리하여 파케트는 8 또는 16, 40, 136개의 기호들을 가질수 있다.

원천마디가 목적마디로부터 자료를 읽으려 한다고 하자.

원천(요청기라고 한다.)은 요청파케트를 요청송신대기렬을 거쳐 SCI고리에 주입한다. 요청의 머리부는 주소와 명령, 다른 정보를 포함한다. 이 파케트는 목적지마디에 도착할때까지 고리를 따라 마디를 차례로 통과한다.

어떤 마디가 목적지라는것을 어떻게 알수 있겠는가?

마디의 스트립퍼(Stripper)는 들어 오는 파케트들속에서 주소가 그 마디의 ID와 일치하는것을 꺼내서 수신대기렬들중의 하나에 넣는다. 목적마디(응답기라고 한다.)는 읽기요청을 해석한후 자료를 꺼내서 응답파케트를 형성한 다음 그것을 SCI고리에 주입한다. 다음 응답파케트는 요청파케트와 꼭같이 고리를 거쳐 원천마디로 간다.

파케트가 목적지가 아닌 마디(말하자면 마디 A)에 도착하면 다음마디로 가는 SCI출력련결로 곧바로 간다. 그러나 련결이 마디 A자체의 자료전송에 의해 점유되여 있을수 있다. 그러면 파케트는 대피FIFO에 넣어 지며 출력련결이 리용가능하게 될 때까지 기다린다.

SCI박자화

SCI는 론리적으로 동기적이며 모든 련결 및 대면부회로들은 같은 박자(보통 500MHz)로 구동된다. 서로 다른 마디가 다양한 박자들을 처리하기 위하여 수신마디는 파케트들에 넘기기기호들을 삽입 및 삭제하는데 신축성완충기를 사용한다.

송신마디는 전송할 파케트가 없을 때 수신동기화를 유지하기 위하여 넘기기기호들을 보낸다. 후에 송신마디가 파케트를 보낼 때는 이써네트에서와 같이 먼저 동기화서문을 보내지 않아도 된다.

SCI통신규약은 망의 공정한 사용을 담보하기 위하여 혼잡과 교착을 피하는데 주의를 돌렸다. 한가지 실례는 요청과 응답에 분할된 송신/수신 대기렬을 사용하는것이다. 분할된 대기렬들이 없으면 련속적인 요청들이 응답의 송신을 억제하여 혼잡에 빠지게 된다.

실례 6.11. Data General **SCI접속다중봉사기클라스터**

Data General SCI다중처리기구조를 그림 6-39에 제시한다. 2개의 SCI고리들이 4개의 SMP봉사기마디들을 호상접속하는데 사용된다.

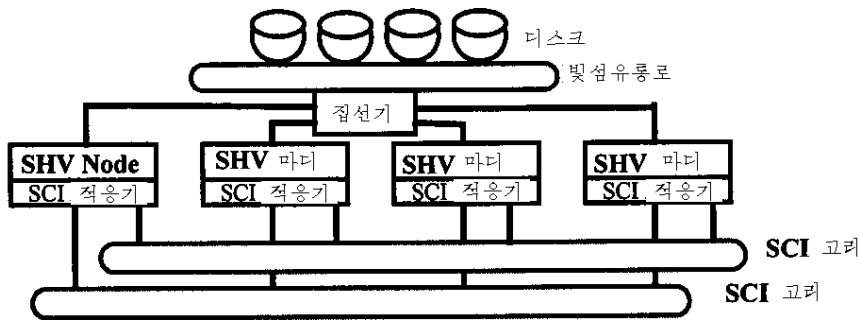



그림 6-39. 2중 SCI 고리들을 가지는 자료일반 CC-NUMA 클라스터

매 봉사기마디는 4GB의 국부기억기를 가지는 4개의 Intel의 상품 Pentium Pro SHV(standard high volume)처리기를 가진다. 2중SCI고리들은 8개까지의 SHV마디들을 지원하도록 총 1GB/s의 대역너비를 제공한다. 이 클라스터는 확대가능한 자료기지응용들을 위한 대규모기업봉사기를 구성하기 위하여 설계되였다. 모든 분산기억기들은 캐쉬일관성 NUMA모형에 따라 전역일관성기억기(Global Coherent Memory, GCM)를 형성할수 있다. Data General SCI설계는 총 32GB의 DSM을 가지는 8개까지의 SHV봉사기마디들을 지원할수 있다. 빛섬유통표고리는 공유디스크렬들을 집선기교환기를 거쳐 모든 봉사기마디들에 접속하는데 사용된다.

이 체계는 분산일관성캐쉬들을 장려하여 CC-NUMA구조와 SCI호상접속적응기를 지원하는 DG/UX Unix조작체계속성들을 개발한다.

체계는 16개의 Pentium Pro처리기로 된 4개의 SHV마디들에 OLTP와 같은 작업부하를 줄 때 초당 14000작용의 속도를 가진다. 이것은 상품처리기들과 공유기억기, 열린 SCI구조를 가지는 SMP봉사기설계의 전형적인 실례이다.

6. 6. 3. SCI일관성규약

SCI에 사용된 캐쉬일관성규약들은 등록부(directory)에 기초한다. 공유목록은 참조를 위하여 등록부들을 하나로 련쇄시키는데 사용된다.

공유목록구조

SCI에서 공유목록들은 캐쉬일관성의 사용을 위해 련쇄된 등록부들을 구성하는데 사용된다. 공유목록들의 길이에는 한계가 없다. 공유목록들은 동적으로 창조되고 다듬어지며 파괴된다.

매개 일관성캐쉬블로크는 블로크를 공유하는 처리기들의 목록에 들어 간다.

공유처리기들사이의 통신은 그림 6-40에서와 같이 공유기억기조종기에 의해 지원된다.

처리기들은 국부적으로 캐쉬화된 자료를 위하여 일관성규약들을 우회하는 선택도 가진다. SCI는 등록부들을 분산시킴으로써 중앙등록부의 사용으로 인한 확장제한을 피한다.

다른 블로크들은 국부적으로 캐쉬되며 일관성규약에 맞지 않는다. 모든 블로크 주소에 대하여 기억기와 캐쉬의 입력자료들은 공유목록에서 첫번째 처리기(머리)를 식별하며 이전의 마디와 다음의 마디들을 련결하는데 리용되는 추가적인 꼬리비트들을 가진다.

매 련결에서 2중화살로 보여 준 앞쪽 및 뒤쪽 지시자를 가지는 2중련결목록들은 공유목록들안의 처리기들사이에서 유지된다. 비일관성의 복사는 페지수준의 조종에 의해 일관성으로 만들수도 있다. 그러나 그러한 더 높은 수준의 쏘프트웨어일관성규약은 공개된 SCI표준의 범위를 벗어 난다. 뒤쪽 지시자들은 련결된 목록의 중간에서 입력자료들에 대한 독립적인 삭제를 지원한다.

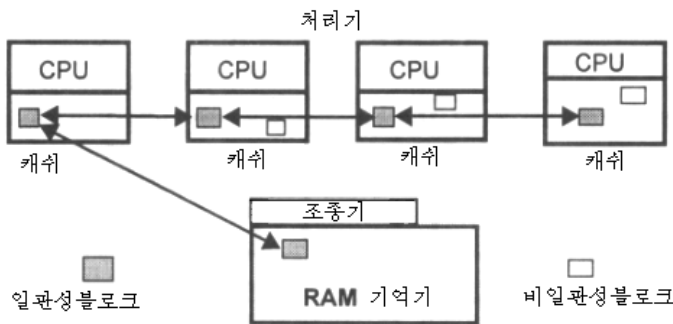



그림 6-40. 분산된 등록부들을 가지는 SCI캐쉬일관성규약

공유목록창조

공유목록의 상태는 기억기의 상태와 목록입력자료들의 상태에 의해 결정된다.

보통 공유기억기는 홈상태(캐쉬되지 않은 상태)에 있거나 캐쉬(공유목록)상태에 있다. 공유목록의 입력자료들은 다중입력자료공유목록에서의 입력자료위치를 서술하는데 목록

에서 입력자료만을 식별하거나 clean이나 dirty, valid, stale과 같은 입력자료의 캐쉬속성들을 서술한다.

머리처리기는 목록관리의 책임을 진다. 기억기와 입력자료상태들의 안정되고 합리적인 결합은 캐쉬되지 않은 자료와 여러가지 위치의 깨끗한 또는 불결한 자료, 캐쉬쓰기가능 또는 변질된 자료를 서술할수 있다. 기억기는 초기에 홈상태에 있으며 모든 캐쉬복사가 타당치 않다. 공유목록창조는 입력자료가 비타당에서 미정상태로 변화되는 캐쉬에서 시작한다.

처리기로부터 기억기조종기로 캐쉬읽기작용이 명령되면 기억기상태는 캐쉬되지 않는 상태에서 캐쉬된 상태로 변화되여 요청된 자료가 돌려 진다. 그다음 요청자의 캐쉬입력자료상태는 미정상태에서 깨끗한 상태(only−clean state)로 변한다. 여러 요청들이 동시에 생성될수 있지만 그것들은 기억기조종기에 의해 순차적으로 처리된다.

공유목록갱신

다음의 기억기접근을 위하여 기억기상태는 캐쉬된다. 공유목록의 캐쉬머리는 불결한 자료를 가질수 있다. 새 요청자(캐쉬 A)는 처음에 기억기에 자기의 캐쉬읽기작용을 명령하지만 요청한 자료대신에 캐쉬 B에로의 지시자를 수신한다.

두번째 캐쉬 대 캐쉬작용은 캐쉬 A에서 캐쉬 B로 명령된다. 다음 캐쉬 B는 자기의 뒤쪽 지시기를 캐쉬 A로 설정하고 요청한 자료를 되돌린다. 임의의 공유목록입력자료는 목록에서 자체로 지울수 있다. 그러나 가능한 혼잡을 피하기 위하여 새로운 공유목록입력자료들의 추가는 선입력선출력으로 수행되여야 한다.

공유목록의 머리는 독점적인 입력자료를 얻기 위하여 다른 입력자료들을 지우는 권한을 가진다. 다른것들은 새로운 목록머리부로서 다시 넣을수 있다. 지우기는 순차로 수행된다. 련쇄된 등록부일관성규약들은 작용들이 없어 질 때 불결한 자료들이 절대로 지워 지지 않도록 한다.

6. 7. 망기술들의 비교

여기서는 앞부분들에서 고찰한 여러가지 망기술들을 비교한다. 먼저 망구조와 케블매체, 규약표준들을 비교한다. 그다음 속도와 케블거리, 대표적인 응용들에서의 상대적인 성능들을 비교한다.

6. 7. 1. 표준망들과 전망

망의 구조들과 표준들, 앞으로의 전망을 아래에 제시하였다.

망표준들

표 6−8에서는 매체공유구조와 교환식구조에 따라 망을 분류한다.

표 6-8 망구조와 케블매체, 표준들

기술	기가비트 이써네트	HiPPI	빛섬유통로	FDDI	SCI	ATM	Myrinet
구조	매체공유, 교환식	교환식	매체 공유, 교환식	매체 공유	매체 공유	교환식	교환식
매체형태	UTP, 동축 케블, 빛섬유	빛섬유, 50쌍 STP	STP, 동축 케블, 빛섬유	빛섬유, 동선	STP, 빛섬유	UTP, STP, 빛섬유	전기적접속, 판빛섬유
표준화단체	IEEE802.3z, Gigabit 이써네트 Alliance	ANSI X3.183, X3.210, X3.218, X3.222	ANSI X3T11	ANSI X3T9.5	IEEE 1595-1992	ATM, Forum, EETF ITU-TSS	Myricom
UTP:비차페쌍꼬임선, STP:차페쌍꼬임선							

케블매체는 동선과 빛섬유케블이다. 규약들과 표준들은 ANSI나 IEEE, Alliance, Forum과 일치된다.

앞으로의 전망

고속이써네트와 기가비트이써네트는 잘 확립된 이써네트표준과의 뒤쪽 호환성으로 하여 LAN과 인트라네트시장을 지배하는 가장 높은 능력을 가지고 있다. ATM기술은 원격통신공업에 의해 힘들게 유지되였고 아직까지 유지되고 있다. ATM은 특히 인터네트망화와 인터네트, 다매체응용들에 적합하다. 앞으로 ATM제품의 생산원가가 낮아 지면 MAN과 LAN령역에 침투할수 있다.

MAN과 WAN응용들에서 ATM과 FDDI는 기본적인 후보자들이다. 10BaseT이써네트와 빛섬유통로, HiPPI는 캐블거리상에서 다음자리를 차지한다. ATM기술의 LAN응용에로의 진출은 이써네트제품들보다 높은 가격으로 하여 제한된다.

오늘 대부분의 ATM망들은 25Mbps~155Mbps의 낮은 속도에 있다.

원인은 ATM련결들과 교환기들, 적응기들의 값을 더 낮춘데 있다. 실례로 다중방식빛섬유를 가지는 622Mbps ATM망의 설비가격은 포구당 6600$로서 1996년의 교환식고속이써네트(포구당 785$)보다 8.4배 더 높다. ATM과 기가비트이써네트의 더 높은 부가원가는 그것들의 앞으로의 장성을 제한할것이다.

SCI와 HiPPI, Myrinet, 기가비트이써네트, 빛섬유통로는 모두 Gbps의 SAN과 LAN, 클라스터응용들을 위한 훌륭한 후보자들이다. 특별한 체계호상접속의 선택은 가격/성능의 절충에 따라 결정된다. 일반적인 경향은 망들을 매체공유의 사용으로부터 교환식구조로 갱신하는것이다. 빛섬유케블은 높은 속도뿐아니라 먼 거리로 하여 전망이 좋다. 최종적인 선택은 흔히 성능보다는 오히려 예산에 의해 결정된다.

6. 7. 2. 망의 성능과 응용

아래에서 주요망기술들의 대역너비(속도)와 지연, 케블거리를 비교한다. 이 기술들은 클라스터들 또는 확대가능한 체계들을 구성하는데 리용되였다. 주어 진 모든 수값들은 Mbps 또는 Gbps에서의 가장 빠른 속도에 대응한다.

동선(UTP 또는 STP)을 리용하거나 한파모습 또는 다중방식의 빛섬유케블을 사용하는데서 가장 먼 거리는 m 또는 km로 표현된다. 표에 기입된 값들은 망기술이 급속히 변하고 있으므로 앞으로 변화된다. 표 6-9는 1997년까지 공개된 망의 속도와 거리성능을 개괄하고 있다.

성능스펙트르

미래의 망들에서 기가비트속도는 초보적인 요구로 될것이다. 이것은 인터네트와 인트라네트, 다매체, 상업응용들에서 훨씬 더 높은 대역너비에 대한 요구가 늘어 나기때문이다. 다중콤퓨터의 클라스터들 또는 확대가능한 다중처리기들에서 상업망의 사용은 급속히 늘어 났다. 앞으로 테라flop/s의 콤퓨터체계들을 지원하기 위하여 테라bit/s의 망들이 필요하다.

표 6-9 망성능과 케블거리

기술	대역너비	동선거리	한파모습 및 섬유거리	다중방식빛섬유거리
기가비트 이써네트	1Gbps	25-100m	2km	500m
HiPPI	800Mbps,1.6,3.2Gbps,SuperHiPPI에서는 6.4Gbps	25m	빛섬유확장기로 20km	직접련결 300m, 확장기로 10km
빛섬유 통로	100,200,400,800Mbps, 1,2,4Gbps	100Mbps에서 50m	10km	200Mbps에서 2km
FDDI	100Mbps,200Mbps	100m	60km	2부터 200km 까지
100 Base-T	100Mbps	100m	20km	반2중에서 412m, 완전2중에서 2km
100VG-AnyLAN	100Mbps	150m	2km	직경 4km
ATM	25Mbps부터 51,155,622Mbps까지	인터네트망화접속을 위한 반복기 또는 빛섬유확장기 사용에 따르는 가변거리		
SCI	800Mbps 부터 8Gbps까지	10m	경험에 의해 100m이상	
Myrinet	1.28Gbps	25m	알려 지지 않음	50m

기가비트모선/교환상 접속들은 많은 상업봉사기들과 대형콤퓨터들, 클라스터들, MPP들에서 출현하였다. 기가비트속도는 SCI와 HiPPI, 빛섬유통로, Myrinet, 기가비트이써네트기술로 구성된 망들에서 실현된다.

상업ATM과 FDDI, 고속이써네트, IsoEnet, 이써네트망제품들은 경제적인 리유로 하여 아직 10Mbps에서 100 또는 622Mbps까지의 낮은 속도범위로 제한된다. SAN과 LAN응용에는 모선과 교환기, 망기술들모두가 적합하다.

특수한 망의 선택은 주어 진 응용에 대한 가격/성능비의 설정에 의존한다.

접속경로를 설정하는데서 지연은 거의 쏘프트웨어부가처리로 인한것이며 교환기들과 망들을 거치는 경로정하기통보문들에서의 하드웨어지연은 없다. 표 6-9에 기입된 수값들은 망기술이 빨리 개선되므로 앞으로 변할수 있다.

응용능력

우의 고찰에 기초하여 주요망기술들의 응용능력을 명백히 한다.

- 빛섬유통로는 기억과 망화, 이질의 호스트들과 주변장치들사이의 자료전송을 위한 여러가지 통로와 망표준들과 함께 동작하는 5개의 FC규약층에서 자료를 넘기기한다.
- 주로 낮은 속도를 가지는 매체공유표준이지만 더 높은 믿음성과 유연성을 가진다. 다른 표준과 교환식FDDI호환성을 위하여 많은 속도증가가 필요하다.
- 기가비트이써네트는 잘 확립된 이써네트와 봉사기구조, 자료기지, 창고업무응용들의 목표로 된다. 고속이써네트사용자그룹들로 하여 개선되고 있는 인트라네트와 다매체, 큰 화상파일들, LAN, 그룹계산, 의뢰기.
- Myrinet는 지금 주로 SAN과 클라스터응용들에 적용되고 있다. 그러나 대면부및 규약장벽들이 공통의 표준들에 의해 완화되면 기가비트 LAN분야에 침투할수 있다.
- HiPPI와 SuperHippI는 대형콤퓨터들과 SMP봉사기들, 초고속콤퓨터들사이의 대량자료이동을 위한 고속매체로 계속 사용될것이다. 규약독립성으로 하여 HiPPI는 인터네트망응용들에도 적합하다.
- ATM은 WAN인터네트망화와 다매체, 인터네트, 광대역ISDN응용을 계속 지배할것이다. 앞으로의 개발에서는 가격효과성(제품가격의 삭감)을 가져 올것이다.
- SCI는 확대가능한 호상접속체계들 특히 SMP와 NUMA다중처리기 또는 최근에 Sun/Cray, SGI/Cray, HP/Convex, Data General 등에서 발표한 MPP클라스터들속에서 인기를 얻고 있다.

6. 8. 참고문헌주해와 련습문제

Feng은 1981년까지의 호상접속구조 [288]에 대한 개괄을 주었다. Siegel의 책은 다단 호상접속망들을 포함한다[555]. 다중처리기들과 다중콤퓨터들을 위한 망은 Varma와 Raghavendra에서 론의된다[624]. 최근의 호상접속망들은 Duate와 Yalamanchili, Ni의 책에서 취급된다. Hwang은 확대가능한 체계들과 클라스터들을 구성하기 위한 기가비트망들을 평가하였다[328].

호상접속상의 3가지 문제점은 IEEE MicroMagazine 1995.2, 1996.2, 1997.2에서 제기되였다. Myrinet는 Bodenetal에서 서술되였다[561]. SGI POWERpath-2는 [549]에서 서술된다. IBM Vulcan크로스바와 SP2교환기설계는 [592]에서 론의된다. Digital GIGAswitch는 [194]의 Advantage클라스터들과 함께 서술된다.

기억기통로기술은 Gillet와 Kaufmann[268]에서 서술된다. 완전혼합의 응용들은 Stone에서 주어 진다[588]. 오메가망은 lawrite[395]에 의해 제안되였으며 Cedra과제[382]에서 구성되였다. Preparata와 Vuillemin은 CCC구조를 개발하였다[588]. 심장수축배렬의 개념은 Kung과 Leiserson에 의해 개발되였다[385]. 화음고리는 Arden과 Lee에서 취급되였다[43]. k-ary N차원립방체망들은 Dally에서 연구된다[183]. 굵은 나무는 Leiserson이 소개한다[403]. Hwang과 Ghosh는 대규모병렬계산을 위한 hypenet[183]을 제안하였다. Stallings는 [579]와 [580]에서 LAN과 MAN, ISDN에서의 최근의 발전을 개괄하였다. ATM과 세포교환기술은 Partridge의 Gigabit Networking[489]에서 취급된다. 일부 망자료는 Saunder[536]에 의한 고속LAN들에 대한 McGraw-Hill수첩에서 얻을수 있다. Johnson은 망구조들에서 집체통신속성들을 밝혔다[353]. Ni와 Mckinley는 wormhole1경로정하기기술을 개괄하였다[465].

기가비트이써네트에 대한 정보는 [267]과 Roberts[518]에서 찾을수 있다. 빛섬유통로는 Frymoye[257]에서 평가되며 FDDI고리는 [582]에서, HiPPI는 Tolmie와 Flanagan[613]에서, SCI는 Bryhni와 Wu[114]에서 평가된다. SCI규약은 IEEE표준 1596-1992[343]에 명기되였으며 James el al. [348]이 소개한다. Myrinet기술과 제품들은 Boden et al.[97]에 서술된다.

병렬빛섬유 SCI련결들은 Engerbretsenh et al. [231]이 소개한다. ATM Forum UNI 3.1은 [46]에 서술된다. Zegura[664]는 ATM교환기체계설계의 기초적인 문제점들을 취급한다. 여러가지 망표준들과 규약들은 ANSI와 IEEE, ATM Forum, Gigabit 이써네트 Alliance의 출판물들에서 볼수 있다.

문 제

문제 6.1. 호상접속망에 대한 다음의 기본적인 술어들을 정의하시오.

(1) 마디차수
(2) 망직경
(3) 2등분대역너비
(4) 정적접속망
(5) 동적접속망
(6) 비차단식망
(7) 집단내 방송과 방송
(8) 그물과 고리면
(9) 망의 균형성
(10) 다단망
(11) 크로스바교환기
(12) 세포교환망

문제 6. 2. 3차원고리면과 6차원2진하이퍼립방체, 최소직경을 가지는 CCC들을 사용하여 64개의 마디로 된 다중콤퓨터를 설계한다고 하자.

아래에서는 이 망위상들에 대한 상대적인 척도를 제기한다. 대답은 해석적인 리유에 기초해야 한다.

(1) d는 망의 차수, D는 망의 직경, L은 망의 전체 련결수라고 하자. 망의 질은 $(d \times D \times L)^{-1}$에 의해 평가된다. 이 질평가에 따라 3가지 구조를 정렬하시오.

(2) 마디호상간 평균거리는 두 마디사이의 가장 짧은 경로의 hop(련결)들의 수이다. 이 평균은 모든(원천, 목적지) 쌍들에 대하여 계산한다. 한 마디가 거리 i를 가지는 다른 모든 마디들에 통보문을 보낼 가능성이 $D-i+1$이라고 할 때 마디호상간 평균거리에 기초하여 3가지 구조를 정렬하시오.

문제 6. 3. 모두가 $N_0, N_1, N_2, \cdots, N_{63}$으로 표식된 64개의 마디를 가지는 Illiac그물(8×8), 2진하이퍼립방체, barrel shifter를 고찰하자. 모든 망련결들은 쌍방향이다.

(1) 3가지 망의 매개에 대하여 마디 N_0으로부터 정확히 3걸음에 도달가능한 마디들을 모두 찾으시오.

(2) 매 경우에 임의의 마디 N_i로부터 다른 마디 N_j로 자료를 보내는데 필요한 경로정하기걸음의 최소수의 웃한계를 지적하시오.

(3) 1024개의 마디를 가지는 큰 망에 대하여 (2)부분을 반복하시오.

문제 6. 4. n개의 처리기와 m개의 공유기억기모듈을 가지는 다중처리기체계를 구성하는데서 모선과 크로스바교환기, 다단망들을 비교하시오. 단어길이는 Wbit이고 2×2교환기들을 사용한다고 가정한다. 비교는 다음의 4가지 종류의 매개에 대하여 각각 진행한다.

(1) 교환 또는 중계, 선들, 접속기, 케블요구와 같은 장치복잡성들

(2) 처리기와 주기억사이의 단위자료전송의 최소지연

(3) 매 처리기들의 리용가능한 대역너비범위

(4) 치환, 자료방송, 차단처리 등과 같은 통신능력들

문제 6. 5. 다단망에 대한 다음의 물음에 대답하시오.

(1) 방송과 치환을 다 포함할 때 4×4교환기모듈에는 얼마나 많은 정당한 상태들이 있는가? 리유를 들어 설명하시오.

(2) 여러 단으로 4×4교환기모듈들을 사용하는 64개 입력 오메가망을 구성하시오. 차단이 없이 망을 통과하는 단일통과에서 얼마나 많은 치환이 직접적으로 실현될수 있는가?

(3) 1통과치환들의 수와 망을 거치는 하나 또는 그이상의 통과에서 달성가능한 총 치환들의 수는 얼마인가?

문제 6. 6. k-ary *n*차원립방체망에 대한 다음의 물음에 대답하시오.

(1) 얼마나 많은 마디들이 있는가?
(2) 망직경은 얼마인가?
(3) 2등분대역너비는 얼마인가?
(4) 마디차수는 얼마인가?
(5) k-ary *n*차원립방체망과 고리, 그물, 고리면들, 2진*n*차원립방체, 오메가망사이의 그라프리론적인 관계를 밝히시오.
(6) 관습적인 고리면과 접힌(folded) 고리면사이의 차이를 밝히시오.
(7) 고정된 선(wire)의 2등분을 가정할 때 왜 낮은 차원의 망(고리면)들은 낮은 지연을 가지며 높은 차원의 망(하이퍼립방체)들보다 더 높은 분쟁처리량을 가지는가?

문제 6. 7. 아래에 제시하는것은 다음의 가정하에서 4개의 처리기기판과 16개의 기억기기판을 가지는 공유기억다중처리기를 위한 뒤쪽 판(backplane)모선의 설계명세서의 실례이다.

- 모선박자속도=20MHz
- 기억기단어길이=64b ; 처리기는 4개 단어로 된 블로크자료를 요구한다.
- 기억기접근시간=100ns
- 공유된 주소공간=240단어
- 뒤쪽 판에서 리용가능한 신호선의 최대수는 96이다.
- 동기적인 시간화(timing)규약
- 완충기와 넘기기지연은 무시한다.

(1) 최대모선대역너비
(2) 유효모선대역너비(최악의 경우)
(3) 중계방식
(4) 매 신호선의 이름과 기능
(5) 뒤쪽 판에 필요한 확장홈의 수

문제 6. 8. 8×8크로스바교환기들을 사용하여 512개 입력다단오메가망을 구성한다고 하자. 이 망에는 몇개의 단이 필요한가? 망을 구성하는데 몇개의 크로스바교환기가 필요한가? 망을 4096개 마디로 확장하면 몇개의 8×8크로스바교환기가 필요한가?

문제 6. 9. 질적 및 량적설명을 주면서 매체공유망, 파케트교환망, ATM교환망에 대한 다음의 두가지 물음에 대답하시오.

(1) 매체공유망과 교환망사이의 차이는 무엇인가? 매개 망종류에 대하여 망기술의 실례를 2개 들고 그것들의 우점과 부족점들을 밝히시오.

(2) ATM교환망과 전통적인 파케트교환망사이의 주요차이는 무엇인가? ATM교환망의 우점과 부족점을 말하시오.

문제 6. 10. 다음의 표의 빈 칸들을 채우시오. 입력값들은 평가가 진행되는 시간에 따라 변한다는데 주의하시오.

망기술	망구조	대역너비(속도범위)	동선상의 거리	빛섬유상의 거리
기가비트이써네트				
Myrinet				
HiPPI				
SCI				
빛섬유통로				
ATM				
FDDI				

문제 6. 11. SCI기술은 확대가능한 공유기억기다중처리기구성을 위한 대중적인 선택으로 되고 있다. SCI에 대한 다음의 물음에 대답하시오.

(1) SCI련결들로 호상접속된 처리마디들사이에서 분산공유기억기(DSM)를 실현하기 위하여 IEEE SCI표준에서 사용한 캐쉬일관성규약을 설명하시오.

(2) 왜 SCI접속들이 DSM다중처리기들을 구성하는데 채택된 대부분의 매체공유망들보다 더 확대가능한가?

(3) CC-NUMA다중처리기체계의 구성에서 어떻게 SCI련결들을 사용하는지 설명하시오.

(4) 확대가능한 병렬콤퓨터를 만들기 위하여 SCI호상접속을 사용하는데서 현재의 제한과 부족점은 무엇인가?

문제 6. 12. 일리노이즈의 Cedar다중처리기는 그림 6-41에서와 같이 클라스터식오메가망으로 만들어 진다. 단 1에서는 4개의 8×4크로스바교환기가, 단 2에서는 4개의 5×8크

로스바교환기가 사용된다.

클라스터당 8개의 처리기를 가지는 4개의 클라스터로 분할된 32개의 처리기와 32개의 기억기모듈이 있다.

비차단접속을 위한 크로스바교환기사용에서 충돌을 피하기 위한 고정된 우선권도식을 그리시오. 간단히 하기 위하여 처리기로부터 기억기모듈에로의 앞방향접속만을 고찰한다.

(1) 두단이 다 8×8크로스바교환기를 사용한다고 가정하자. 64개 처리기와 64개 기억모듈사이의 교환식접속을 제공하기 위한 두 단계 Cedra망을 설계하시오.
(2) 512개 처리기와 512개 기억기모듈들을 접속하기 위한 블로크를 만드는것처럼 Cedra망을 8×8크로스바교환기를 사용하여 더 확장하시오.
(3) 입력단으로부터 출력단까지 린접한 단사이의 모든 접속들을 말하시오.

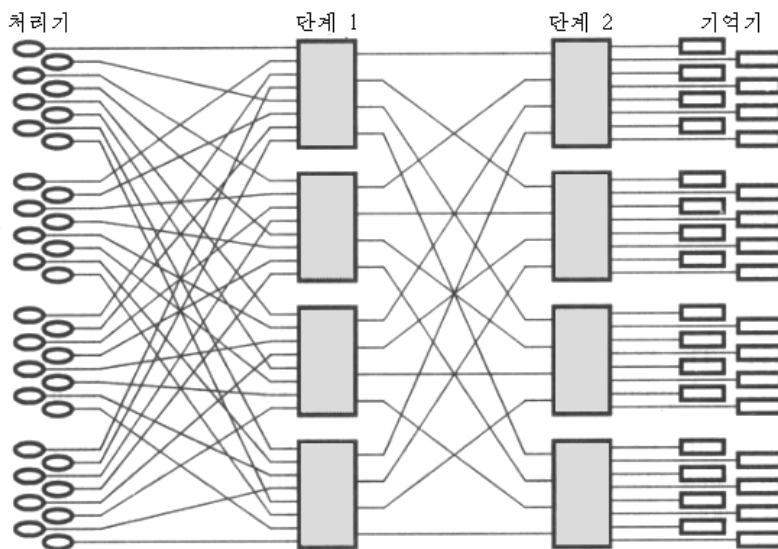



그림 6-41. Cedra 다중처리기안의 오메가망

제7장. 스레드화, 동기화 및 통신

이 장에서는 프로세스/스레드의 관리와 동기화, 통신을 지원하는 여러가지 하드웨어 및 쏘프트웨어기구들을 론의한다. 이 기구들은 3편에서 고찰하는 여러가지 다중처리기와 다중콤퓨터가동환경들에서 사용된다. VLSI기술의 발전으로 이 기구들중의 일부는 단일처리기체계들에 통합되고 있다.

7.1에서는 먼저 2장에서 론의한 프로쎄스의 개념에 대립되는 스레드의 개념을 론의한다. 7.2에서는 여러가지 동기화기구들을 론의한다. 7.3과 7.4은 통신을 취급한다.

먼저 대중적인 TCP/IP표준을 론의하고 다음 보다 효과적인 여러 통신기술을 준다.

7. 1. 쏘프트웨어다중스레드화

하드웨어다중스레드화는 5.6에서 론의한것처럼 기억기지연을 숨기기 위한 다중문맥처리기들의 사용을 의미한다는것을 명백히 한다. 쏘프트웨어다중스레드화는 완전히 다른 개념이다. 현대 처리기구조와 조작체계에서는 두개의 중요한 발전이 이룩되여야 한다.

- 첫째로, 현재의 RISC처리기들을 높은 정수 및 류동소수점연산성능을 가지도록 설계하는것이다. 그러나 조작체계지원만큼 주의를 돌리지 않았다.
- 둘째로, 확대가능한 콤퓨터가동환경들에서 조작체계의 기능성 및 모듈성추가를 위한 보다 구조적인 지원이 필요하다.

이러한 발전은 오늘의 PC시장을 지배하는 Intel 80x86과 Pentium처리기들에도 응용된다. 이 처리기들은 CISC에 기원을 두고 있으며 RISC의 많은 착상을 통합하였다. Andersonelal[36]은 처리기구조와 조작체계의 요구에 대하여 론의하였다. 그 연구의 일부 결과들을 아래의 표(표 7−1)로 만들었다.

R2000과 SPARC는 VAX의 계산속도(SPECmark성능평가기준으로 측정된다.)를 4배 더 개선하였지만 조작체계기능을 위한 속도는 훨씬 적게 개선하였다.

처리기구조와 조작체계사이의 이 차이는 그것을 제공하는 처리기와 조작체계설계자들이 다 관심을 돌리지 않는다면 더 커지게 될것이다.

최근의 조작체계연구에서의 주요한 혁신은 스레드이다. 이 부분에서는 스레드가 어떻게 효과적으로 실현될수 있는가를 본다. 스레드를 다중스레드화된 프로그람작성에 어떻게 리용하는가는 12장에서 론의한다.

표 7-1 처리기구조에 대한 조작체계지원

속 성		절 대 값			CVAX와 상대값	
		DEC CVAX	MIPS R2000	SUN SPARC	MIPS R2000	Sun SPARC
OS 기능 성능(μs)	Null 체계 호출	15.8	9	15.2	1.8	1
	트랩	23.1	15.4	17.1	1.5	1.4
	페지표 입력자료변화	8.8	3.1	2.7	2.8	3.3
	문맥절환	28.3	14.8	53.9	1.9	0.5
OS 기능명령수	Null 체계 호출	12	84	128	7	10.6
	트랩	14	103	145	7.4	10.3
	페지표 입력자료 변화	11	36	15	3.3	1.4
	문맥절환	9	135	326	15	33.2
처리기상태에서 32bit 단어의 수	등록기들	16	32	136	2	8.5
	F.P.상태	0	32	32	-	-
	Misc.상태	1	5	6	5	6

7. 1. 1. 스레드개념

여기서는 IEEE POSIX Threads(Pthreads)[345]와 많은 공통성을 가지며 널리 사용된 상업제품인 Solaris thread[583]에 기초한 단순화된 모형을 사용한다.

스레드(또는 조종스레드)는 실행되고 있는 명령들의 렬이다. 스레드의 개념을 그림 7-1에서 설명한다.

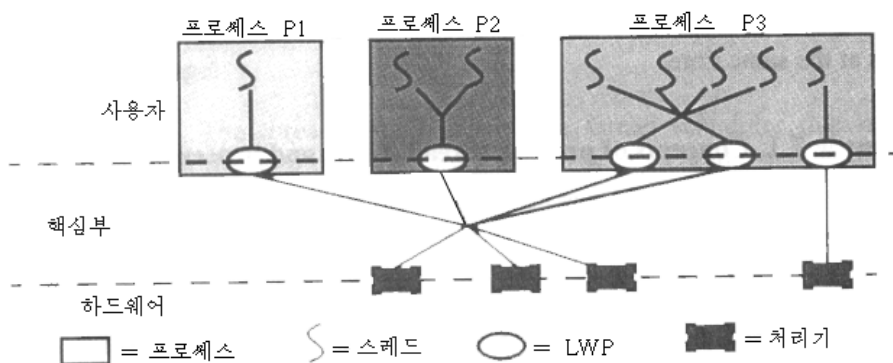



그림 7-1. Sun Solaris 조작체계에서의 다중스레드화

Solaris스레드모형의 기본형태는 2개 수준구조이다. 사용자방식(사용자공간)에서 동작하는 스레드는 사용자수준스레드(또는 사용자스레드 또는 간단히 스레드)라고 한다. 핵심부방식(Kernel mode)에서 동작하는 스레드는 핵심부스레드 또는 경량처리(Light weight process, LWP)[1]라고 한다. 사용자수준의 동적련결스레드서고는 스레드응용프로그람작성대면부를 실현하며 사용자스레드들을 관리한다. 한편 핵심부는 LWP들을 관리한다.

매개 프로쎄스는 하나 또는 그이상의 스레드를 가지며 하나 또는 그이상의 LWP들에 배정된다. LWP를 가상CPU로 생각할수 있다. 하나의 프로쎄스에 배정된 LWP들의 수를 프로쎄스의 **동시성수준**이라고 한다. 스레드서고는 핵심부가 물리처리기들의 련합(pool)우에서 실행하는 모든 프로쎄스들의 LWP들을 일정작성하는것과 꼭같이 LWP들의 련합우에서 실행하는 프로쎄스의 스레드들을 일정작성한다. 핵심부는 사용자수준스레드들을 관리할수 없으며 LWP들만이 관리할수 있다.

하나의 프로쎄스내의 LWP들의 련합우에서 실행하도록 일정이 작성되는 스레드를 **비속박스레드**라고 부른다. LWP에 영구적으로 속박된 스레드는 **속박스레드**라고 한다. 프로그람작성자의 관점에서 속박스레드는 비속박스레드와 꼭 같다. 그러나 조작체계의 관점에서 속박스레드는 LWP와 같다. 실례로 비속박스레드가 프로쎄스내에서 일정이 작성되는 동안 속박스레드는 모든 능동스레드들에 관하여 일정이 작성되는데 이것은 실시간응용들을 위한 유용한 특징이다.

스레드는 프로쎄스와 꼭같이 임의의 응용코드와 체계호출들을 실행할수 있다.프로쎄스의 스레드들은 프로쎄스의 주소공간 즉 코드와 대부분의 자료, 프로쎄스서술자정보를 공유한다. 스레드들은 서로 독립적으로 비동기적으로 실행한다. 하나의 스레드가 차단될때(실례로 완료하기 위하여 I/O연산을 기다리는것) 다른 스레드들은 계속 실행할수 있다. 한개 프로쎄스의 여러 스레드들은 같은 시간에 여러 처리기들에서 실행할수 있다.

실례 7.1. 프로쎄스와 스레드, LWP, 처리기의 개념

그림 7-1의 체계는 4개의 처리기에서 실행하는 3개의 프로쎄스를 가진다. 프로쎄스 P1는 단일스레드와 단일LWP를 가지는 일반적인 유닉스처리이다. 프로쎄스 P2은 단일 LWP우에서 실행하는 2개의 스레드를 가진다. 프로쎄스 P3은 3개의 LWP에서 실행하는 5개의 스레드를 가진다. P1, P2, P3의 동시성수준은 각각 1, 1, 3이다. P1의 스레드와 P3의 제일 오른쪽 스레드는 속박스레드이다. 나머지는 모두 비속박이다.

스레드연산들은 사용자공간에서 스레드서고에 의해 실현되고 프로쎄스안의 스레드들은 분할된 주소공간들을 가지지 않기때문에(경량스레드) 프로쎄스연산들보다 비용이 낮다. 그러므로 창조, 끝내기, 동기화와 같은 스레드연산들은 모두 핵심부에 들어 가지 않고 같은 사용자공간에서 진행된다. 복사와 문맥절환, 교차보호경계들로 하여 프로쎄스의 연산에서 초래되는 부가처리들은 크게 감소된다.

---

[1] 핵심부스레드와 LWP는 2개의 서로 다른 개념이다. 간단화를 위하여 LWP는 사용자의 처리를 실행하는데 리용되는 핵심부스레드로 간주한다. 핵심부스레드들은 새치기처리와 같은 다른 목적에 사용된다. 다르게 서술하지 않는 한 핵심부스레드와 LWP사이의 차이를 이 책에서는 무시한다.

7. 1. 2. 스레드관리

스레드창조(thread creation)

스레드창조는 새로운 주소공간을 창조할 필요가 없기때문에 프로세스창조보다 더 간단하다. 다만 다음과 같은 상태정보가 특정한 스레드에 대하여 유지되여야 한다.

- 스레드ID
- 처리기상태(프로그람계수기와 탄창지시기를 포함하는 등록기들)
- 스레드특정탄창
- 스레드일정작성에 사용되는 스레드의 우선권
- 스레드특정자료를 보관하기 위한 스레드국부기억구역
- 스레드특정신호마스크

이 정보는 스레드구조라고 하는 자료구조에서 스레드서고에 의해 창조되고 유지된다. 탄창용기억기구역은 스레드서고에 의해(프로세스의 주소공간의 더미구역에) 자동적으로 배정되거나 스레드의 창조때 응용코드에 의해 주어진다.

프로세스안의 모든 변수들은 그 프로세스의 모든 스레드들이 공유한다. 그러므로 한 스레드가 변수를 수정하면 그 효과는 그 프로세스의 모든 스레드들에서 나타난다. 또한 매 스레드는 이 특별한 스레드에 소속된 변수들을 보관하는데 사용되는 스레드국부기억도 가지는데 이것에는 다른 스레들이 직접 접근할수 없다.

스레드구조자체도 스레드국부기억에 배정된다.

스레드일정작성

스레드서고는 LWP들의 련합우에서 실행하는 스레드들을 다중화하는 스레드일정작성을 제공한다. 매 스레드는 그림 7-2에서 보여 주는바와 같이 임의의 시각에 어떤 하나의 상태에 있다. 새로 창조한 스레드는 RUNNABLE상태에서 시작하여 휴식 LWP가 리용가능하게 되기를 기다린다. 다음서고는 그것을 급송하여 LWP에서 실행하며 스레드는 ACTIVE상태에 들어 간다.

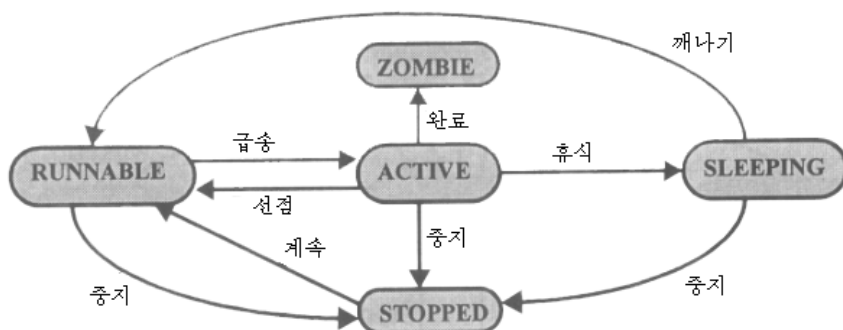



그림 7-2. 스레드서고에 의해 조종되는 스레드상태변화도

ACTIVE스레드가 처리기국부동기화대상(즉 다른 프로쎄스들사이에서 공유되지 않는 것)에서 차단되면 LWP는 그것을 SLEEPING상태에 넣은 다음 가장 높은 우선권을 가지는 다음의 RUNNABLE스레드실행에로 절환한다. 그러나 스레드가 속박스레드이면 스레드의 차단은 밑에 있는 LWP도 차단한다. 잠자기스레드는 동기화대상이 리용가능하게 될 때 즉 차단조건이 제거될 때 깨여 난다.

Solaris스레드들은 선점(preemption)를 지원한다. 더 높은 우선권을 가지는 스레드는 ACTIVE스레드를 선점하여 ACTIVE스레드의 LWP를 가질수 있다. 프로그람작성자는 thr_suspend()와 같은 서고함수들을 사용하여 스레드를 중지하고 그것을 STOPPED상태로 보낼수 있다. STOPPED상태의 스레드는 살아 있지만 thr_continue()가 호출될 때까지 RUNNABLE로 될수 없다. 최종적으로 스레드를 완료하면(스레드코드가 끝나거나 thr_exit()가 실행될 때) 그것은 ZOMBIE상태에 들어 간다. 스레드서고는 이 ZOMBIE스레드들의 기억공간을 해방하는 수확스레드(reaper thread)를 가진다.

ACTIVE스레드는 LWP우에서 실행하도록 미리 급송된 스레드이다. 이 스레드는 아래에 있는 LWP에 련결된다고 한다. 매 LWP는 임의의 주어 진 시간에 최대로 하나의 련결된 스레드를 가진다. ACTIVE스레드의 밑에 있는 LWP는 그림 7-3에서와 같이 임의의 시각에 4개 상태들중 하나에 있다. LWP는 스레드가 다른 스레드상태로부터 또는 다른 스레드상태에로 이행할 때 항상 RUNNING상태에 있다는데 주의하여야 한다.

LWP들은 핵심부에 의하여 일정이 작성된다. RUNNING LWP는 핵심부에 의해 선점되여 있거나 시간초과될 때 RUNNABLE상태로 이행한다. 그것은 LWP가 어떤 조건이 성립될 때까지 기다리도록 차단체계호출을 실행하여 차단할수 있다. LWP는 STOPPED상태에 들어 가는 LWP서고함수호출에 의해 중지될수 있으며 continue서고함수를 호출하여 이전의 상태로 되돌아 갈수 있다.

스레드가 완료되고 RUNNABLE스레드가 더는 없을 때 밑에 있는 LWP는 대역 LWP조건변수를 기다리면서 휴식상태에 들어 간다. 스레드가 RUNNABLE로 될 때 대역변수는 휴식상태의 LWP가 깨나도록 신호한다.

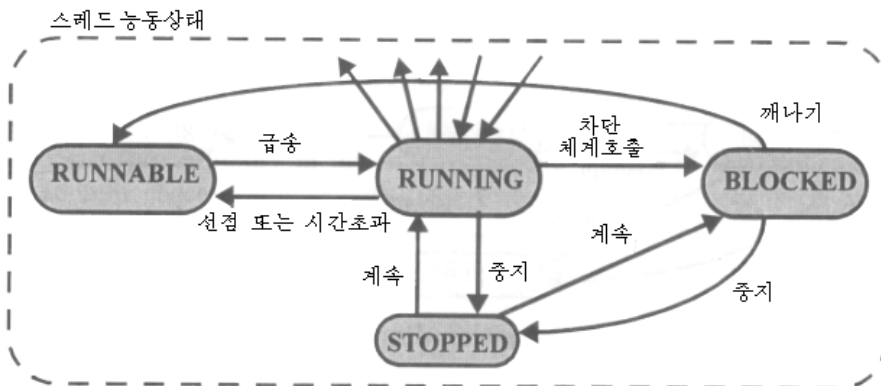



그림 7-3. 핵심부에 의해 조종되는 LWP의 상태변화도

7. 1. 3. 스레드동기화

Solaris스레드서고는 두가지 형태의 동기화대상 즉 프로쎄스국부와 프로쎄스공유를 실현한다. 프로쎄스국부동기화대상은 같은 프로쎄스내의 스레드들에 접근가능하며 같은 프로쎄스의 스레드들을 동기화하는데 사용된다.

스레드서고는 매개 동기화변수에 대한 하나의 잠자기대기렬을 유지한다. 스레드가 동기화변수에 대한 동기화연산을 실행하고 차단하면 스레드는 그 변수의 대기렬에서 잠자기에 들어 간다. 만일 스레드가 비속박이면 밑에 있는 LWP는 다음의 RUNNABLE스레드실행에로 절환된다. 스레드가 속박스레드이면 밑에 있는 LWP도 속박스레드와 련관된 LWP신호기를 기다리면서 차단한다.

이때 그 LWP는 스레드우에 세워 둔다고(park) 한다. 차단된 변수가 리용가능하게 되면 차단된 스레드는 깨나서 RUNNABLE상태에 들어 가며 서고에 의하여 일정이 작성된다. 깨여 난 스레드가 속박스레드이면 LWP는 세워 두지 않고 핵심부에 의해 일정이 작성된다.

프로쎄스공유동기화대상은 여러 처리기들이 접근가능하며 서로 다른 프로쎄스들의 스레드들을 동기화하는데 사용될수 있다. 프로쎄스공유동기화요소들은 핵심부가 차단스레드들의 LWP에 련결되여 있는 동안 차단스레드들을 핵심부안에서 잠자기에 넣기 위하여 LWP동기화요소들에 의거한다.

그러한 대상들은 동기화요소들이 그것들을 프로쎄스공유로 인식할수 있도록 초기화되여야 한다.

7. 2. 동기화기구들

동기화는 많은 콤퓨터부분들에 놓여 있는 풍부한 개념이며 활발한 연구분야이다. 콤퓨터구조와 자료기지, 조작체계, 병렬처리, 프로그람작성언어와 같은 여러 학과들에서 걸보기에 물질과 비슷한 동기화를 왜 배우는가 하면서 놀라는 학생들도 볼수 있다. 동기화학습에서는 4개의 국면을 리해하여야 한다.

호상배제(mutual exclusion), 원자성(atomicity), 생산자-소비자(producer-consumer), 식사하는 철학자들(dining philosophers), 장벽동기화(barrier synchronization) 등과 같은 사용자와 접촉되는 동기화문제들.

(1) Andrew[37]는 여러가지 기타 문제들에 대한 훌륭한 참고서이다. 호상배제와 원자성문제들은 특별히 중요한데 7.2.1에서 고찰한다.
(2) 동기화문제들을 풀기 위하여 사용자가 채용하는 언어구문들.
(3) 구문들은 본래의 구문들이나 서고부분루틴들, 콤파일러지령들의 형식일수 있다. 현재의 공유기억기프로그람작성환경에서 대중적인 구문들은 잠금, 신호기발, 사건, 림계구역, 장벽 등을 포함한다. 이 구문들을 고수준구문이라고 하는데 7.2.2과 12장에서 상세히 론의한다.
(4) 검사와 설정, 꺼내기와 첨가, 비교와 교환 등과 같은 다중처리기구조에서 리용가

능한 동기화요소들.

(5) 이것들은 흔히 체계하드웨어에 의해 직접 제공되므로 저수준구문이라고 한다. 이것들은 7.2.3에서 학습한다.

(6) 저수준구문들로 고수준구문들을 실현하는데 사용되는 알고리듬들. 현재의 많은 체계들에서 잠금들은 여전히 기본적인 구문이며 그우에서 다른 구문들이 만들어진다. 그래서 다음절들에서는 저수준의 구문들을 리용하여 어떻게 효과적인 잠금을 구성하는가 론의한다.

7. 2. 1. 원자성 대 호상배제

2.4.1에서 이미 언급한바와 같이 3가지 형태의 동기화연산 즉 원자성과 자료동기화, 조종동기화가 있다. 보다 상세한 계층도는 그림 7-4에 보여 준다.

현재의 다중처리기에 대한 본래의 프로그람작성모형은 장벽과 호상배제, 잠금/신호기발, 생산자-소비자동기화를 지원한다. 이것들은 원자성 및 eureka동기화를 충분히 지원하지 못한다. 비형식적으로 장벽동기화는 모두가 그 실행에서 하나의 일정한 점에 도달하는 프로쎄스들의 그룹을 담보한다.

Eureka동기화는 장벽동기화의 반대이다. 즉 하나의 프로쎄스가 하나의 일정한 점에 도달하면 즉시에 모든 프로쎄스들에 비동기적으로 통지된다(Eureka! I got it!).

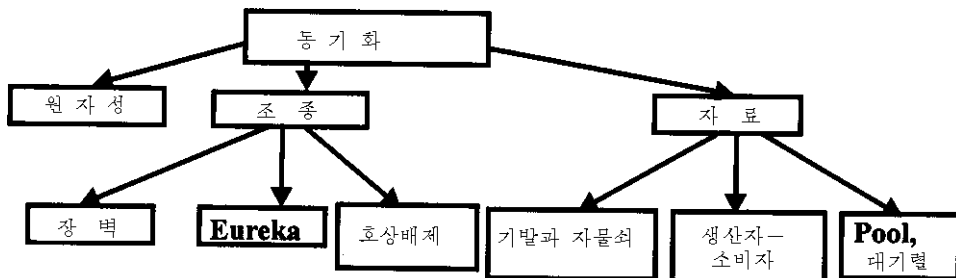



그림 7-4. 동기화의 여러가지 형태들

이것은 병렬탐색과 같은 응용들에서 유용하다.

련합의 동기화는 빈 련합으로부터 항목을 꺼내거나 가득찬 련합에 항목을 넣지 말것을 요구한다. 대기렬에 대한 동기화도 어떤 순서(실례로 FIFO대기렬의 우선권대기렬)가 있다는것을 제외하고는 류사하다. 모든 동기화는 보통 공유기억기기계들(PVP들, SMD들, DSM들)우에서는 일부 잠금요소들, MPP들과 클라스터들우에서는 통보문넘기기요소들에 의해 실현된다.

호상배제와 원자성은 련결되였지만 서로 다른 2개의 개념이다. 호상배제는 간단히 2개의 물건들은 같은 시간에 같은 위치에 있을수 없다는것을 규정한다.

원자성은 연산들의 렬은 한정되고 더 쪼갤수 없는 단일한 걸음(step)으로써 실행되여야 한다는것을 규정한다.

호상배제문제

호상배제문제는 보다 형식적으로 다음의 코드로 서술할수 있다.

```
parfor(i=0;i<n;i++)                         // parfor순환의 매 실행에.
{                                           // n개의 처리기가 있다
    while(Condition){
        independent computation, noncritical section
        critical {                  //호상배제코드의 입력자료점
                critical section
        }                           //호상배제코드의 탈퇴점
        independent computation,noncritical section
    }
}
```

critical section은 호상배제적으로 실행되여야 할 코드의 토막이다. Condition은 critical section에서 수정되는 론리표현이다. 실마리어 **critical**은 괄호 {와 }로 둘러 싸인 코드(critical section)가 호상배제적으로 실행되도록 담보하는 구조화된 언어구문인 critical region(림계구역)을 서술한다. 호상배제문제의 해답은 다음의 3가지 속성을 만족시켜야 한다.

- **호상배제성** 한번에 오직 하나의 프로쎄스만이 critical section을 실행되도록 하여야 한다.
- **담보전진** 만일 여러 프로쎄스들이 자기의 critical section을 수행하는 림계구역에 들어 가려고 시도하면 적어도 하나는 끝내 성공한다. 다시 말하여 프로그람은 혼잡되거나 교착되지 말아야 한다.
- **기아해제** critical section을 실행하려는 프로쎄스의 시도는 성공한다. 그것은 다른 처리기들의 공모에 의해 기아에 빠지지 않는다.

좋은 해답은 역시 효률적이여야 한다. 실례로 호상배제알고리듬의 성능은 전체 프로쎄스들의 수 n에 의존하지 않고 critical section을 실행하려고 경쟁하는 프로쎄스들의 수 m에만 의존하여야 한다. 값 $n-m$은 경쟁하지 않는 프로쎄스들의 수인데 이것들은 자기의 비림계구역코드를 실행하고 있거나 완료하였다.

다른 실례로서 만일 프로쎄스가 critical section을 실행하고 있지 않다면 림계구역에 들어 가려는 프로쎄스의 시도는 즉시적으로 수행되여야 하며 다른 비경쟁프로쎄스들에 의해 지연되지 말아야 한다.

Lamport의 애완용동물문제

호상배제문제의 유명한 표현은 Lamport[390]가 제기하였다.

알리크와 보보는 하나의 마당을 공유하는 이웃들이다. 알리크에게는 한마리의 애완용고

양이가 있고 보보에게는 애완용개가 있다. 그들은 자기들의 애완용동물들을 마당에서 놀게 하려고 한다. 그러나 두 동물은 서로 좋아 하지 않기때문에 매 시각에 오직 한마리만이 마당에 있을수 있다. 그리하여 알리크와 보보는 자기의 애완용동물들이 싸우지 않고 마당에서 놀게 하는 방법을 찾아 내야 한다.

이것은 2개의 프로쎄스문제이며 n개 프로쎄스로 쉽게 일반화할수 있다. 호상배제는 한번에 하나의 애완용동물만이 마당에 있을수 있다는것을 가리키고 있다. 담보전진은 어떤 애완용동물이 마당에서 놀수 있다는것을 가리키고 있다. 기아해제는 애완용동물이 마당에서 놀려고 무한정 기다리지 않는다는것을 가리키고 있다.

원자성

원자연산개념은 상태기계모형을 리용하여 형식적으로 정의할수 있다. 지령적인 프로그람작성에서 프로그람은 상태기계(또는 자동체)로 간주할수 있다. 프로그람은 초기상태로부터 시작한다. 다음 프로그람은 더 쪼갤수 없는 걸음들의 렬을 따라 가는데 매 걸음은 현재의 상태를 다음상태로 변화시킨다. 그러한 더는 쪼갤수 없는 상태변환걸음을 원자연산이라고 부른다. 보다 형식적으로 다음의 정의를 준다.

정의 7.1 프로그람에서 원자연산은 다음의 3가지 속성을 가지는 지령들의 렬이다.

- **유한성** 원자연산의 시작으로부터 끝까지의 시간은 유한이다.
- **불가분리성** 프로그람작성자의 관점에서 원자연산은 단일한 더는 쪼갤수 없는 걸음으로 실행된다. 프로그람의 나머지부분은 원자연산의 최종결과만을 볼수 있다. 원자연산에 의해 생성된 어떤 중간결과도 보이지 않는다. 만일 어떤 리유로 원자연산이 중간에서 중단되면 그것은 모든 부분적인 효과들을 본래대로 만들고 출발상태에로 복구하여야 한다.
- **렬거가능성** 불가분리성의 응당한 결과는 렬거가능성인데 이것은 여러개의 원자연산들이 병렬로 실행될 때 최종결과는 마치 이 연산들이 어떤 임의의 순서로 잇달아 실행된것과 같아야 한다는것을 의미한다.

이 정의에서는 원자연산이 프로그람작성자의 관점에서 정의되였는데 조작의 단위는 지령이다. 콤퓨터구조는 모선주기 또는 박자주기와 같은 보다 상세한 수준에서 원자성을 리용하려고 한다. 보통 콤퓨터의 구조와 프로그람작성환경은 산수/론리연산과 기억기읽기, 기억기쓰기, 검사와 설정, 비교와 교환, 꺼내기와 첨가 등과 같은 일부 기본적인 원자연산들을 제공한다.

원자성문제는 이러한 기본적인 연산들로 어떻게 단일한 원자연산을 만드는가 하는것이다. 그러한 더 큰 원자렬을 **트랜잭션**이라고도 부른다. 원자성문제의 해답은 원자성과 담보전진, 기아해제를 담보하여야 한다. 원자개념은 다음의 예금문제를 통하여 더욱 정교해진다.

예금문제

하나의 큰 회사가 은행에 3개의 구좌 *X*, *Y*, *Z*를 가지고 있다. 이 구좌는 임의의 수의 회사종업원들이 그림 7-5의 실례와 같은 3개의 원자연산을 통하여 접근할수 있다. 임의의 시간동안 많은 종업원들이 이 구좌들에 동시에 접근하고 있다. 이 트랜잭션은 안전하고 효률적이여야 한다.

간단히 하기 위하여 구좌 A가 충분한 자금을 가지고 있지 않을 때 회수와 이동에 필요한 오유처리코드들은 무시한다. 그림 7-5에서 코드들은 원자성을 서술하기 위하여 새로 보충된 구문을 가지는 C-like기호를 사용한다.

이 원자구역구문 "**atomic**{ 구역체부 }"은 예약단어 **atomic**로 시작되며 구역체부의 모든 연산들이 단일원자연산으로 실행되야 한다는것을 서술한다.

원자구역은 일관성구역이라고 하는 보다 강력한 구조화된 구문의 특수한 형식이다. 이것은 12.3.5에서 론의한다.

```
구좌 X로부터 $100 꺼내기 :
    atomic {
        if ( balance[X] > 100 ) balance[X] = balance[X] – 100 ;
    }

구좌 Y로 $100 넣기 :
    atomic { balance[Y] = balance[Y] + 100 ; }

구좌 X로부터 구좌 Y로 $100 옮기기:
    atomic {
        if ( balance[X] > 100 ) {
            balance[X] = balance[X] – 100 ;
            balance[Z] = balance[Z] + 100 ;
        }
    }
```

그림 7-5. 예금문제에서 3개의 트랜잭션

원자성과 호상배제

원자연산들과 호상배제연산들은 다음의 차이를 가진다.

- 원자연산은 유한이지만 유한성이 critical section과 같은 호상배제연산의 내재적인 특성은 아니다. 이 문제는 현재의 병렬언어들에서는 무시된다. critical section은 콤파일러나 실시간지원체계가 발견할수 없는 무한고리를 포함할수 있다.
- 호상배제는 불가분리성을 담보할수 있다. 그러나 불가분리성은 다른 비호상배제수단에 의하여 실현될수 있다.
- 호상배제는 순차적인 실행을 가능하게 하지만 원자성은 병렬실행을 가능하게 한다. 호상배제문제에서는 한번에 *n*개의 프로세스들중 하나만이 critical section을

실행할수 있다. 예금문제에서 여러 트랜잭션들은 원자성이 실시되는 한 동시적으로 실행될수 있다.

원자성은 전통적으로 호상배제에 의해 실현된다. Dijkstra가 신호기발구문을 도입함으로써 잠금은 호상배제, 원자연산들을 지원하는 우월한 기술로 되였다. 원자성에 대한 호상배제적인 잠금해답은 여러개의 동시적으로 실행하는 프로쎄스들이 공유자원들을 놓고 경쟁하는 조작체계들의 설정에서 매우 자연스럽다.

거기에서 기본관심은 안전한 자원공유를 담보하는것이다. 실례로 두개의 프로쎄스가 다 같은 인쇄기로 파일을 인쇄하려고 한다고 가정하자. 이 두개의 인쇄연산들은 량쪽 파일들로부터 혼합된 행들이 출구되지 않도록 동기화되여야 한다. 하나의 프로쎄스가 먼저 인쇄기를 잠그어 자체의 호상배제적인 접근을 허용하고 인쇄연산을 끝낸 다음에만 인쇄기를 해제하는것은 자연스럽다.

병렬계산에서는 다르다. 여기서는 프로쎄스들이 함수를 계산하는데서 협력한다. 자원공유는 프로그람작성자에게서 기본적인 관심사가 아니며 호상배제는 더는 원자성을 실현하는 본질적인 방법이 아니다. 사실 호상배제는 병렬계산의 목적에 대립된다. 왜냐하면 병렬계산은 순차실행을 요구하기때문이다.

그러나 현재의 다중처리기들은 아직도 원자연산에 대한 기본적인 지원으로서 잠금과 림계구역과 같은 호상배제구문을 사용한다. 호상배제는 병렬계산응용들에서 원자성을 실현하는 유일한 또는 가장 좋은 방법이 아니다.

새로운 병렬언어구문들과 구조지원들이 원자성을 더 효과적이고 안전하게 실현하기 위하여 연구되고 있다. 그러나 원자성이 critical section문제로만 모형화될수 있으며 이것은 호상배제를 요구하고 따라서 잠금을 요구한다는 생각을 깨버려야 한다.

실례 7.2. 다시 본 Lamport의 애완용동물문제

- 알리크와 보보는 어느 애완용동물이 어느 시간에 마당을 사용할수 있는가 하는 일정을 작성할수 있다. 사실 SIMD와 PRAM과 같은 동기식병렬콤퓨터들에서 호상배제문제는 없어 진다.
- 알리크나 보보는 다른 이웃으로 옮겨 갈수 있다. 이것은 공유변수가 더는 공유되지 않도록 하는 병렬프로그람의 알고리듬에 대한 수정과 같다.
- 알리크와 보보는 마당에 울타리를 만들어 매 애완용동물이 자기의 마당을 가지도록 할수 있다. 이 해답은 병렬프로그람작성에서 널리 사용된다. 착상은 공유자료구조를 더 작은 쪼각들로 나누어 매 프로쎄스가 경쟁이 없이 자기의 자료쪼각우에서 작업하도록 하는것이다.
- 알리크와 보보는 이웃에 자기들의 애완용동물이 마당에서 놀고 싶어 한다는것을 알기 위하여 풍선을 띄울수 있다. 애완용동물은 이웃의 풍선이 이미 점유되였음을 발견하면 집으로 가서 후에 다시 시도한다. 이 착상은 잠금해제동기화기술[306]에서 제기된다. 해답은 여러가지이다. 두 짐승이 잠간 싸우게 하고 진 놈은 쫓아 낸다. 떠 있지 않으면 마당으로 갈수 있다. 이것은 기발을 리용하는 Lamport의 해답과 같다[390].
- 애완용동물은 아무때나 마당에 갈수 있다. 그러나 마당이 성숙된 술어로 이것은 우선권과 선점을 허용하는것과 같다.

그림 7-6. Lamport의 애완용동물문제에 대한 소녀의 해답

오유를 범하기가 쉽다는것을 설명하기 위하여 Lamport의 애완용동물문제를 다시 본자. 콤퓨터과학자가 이 문제를 풀면 거의나 잠금과 잠금해제(unlock)의 쌍을 사용하는 잠금해답이 얻어 질것이다.

이제 어린이가 같은 문제를 어떻게 푸는가 보자. 10살 난 소녀에게 우의 문제를 풀게 하였다. 그 소녀는 12개의 해답을 제기하였다. 애완용동물들이 마당에 있는 동안 종을 달도록 하거나 휴대용무선전화기를 사용하도록 하는 등 일부 해답들은 실지 타당치 않다. 소녀는 5개의 타당한 해답들을 주었는데 그것들을 그림 7-6에 요약한다.

이 문제가 호상배제문제(원자성문제는 아니다.)로서 소녀에게 의도적으로 제기되였음에도 불구하고 소녀는 열쇠나 림계구역을 사용하는 호상배제해답을 제기하지 않았다는것은 흥미 있다. 그러나 그의 4번째 해답은 그 방향에 있다. 호상배제의 개념 즉 두개의 물건이 같은 시간에 같은 장소에 있을수 없다는것을 아이들에게 설명하기 쉽다는것도 알수 있다. 그러나 원자성개념이나 원자연산의 개념을 설명하기는 힘들다.

7. 2. 2. 고수준동기화구성

현재의 공유기억기다중처리기들에 대한 병렬프로그람작성환경들은 4개 형태의 동기화요소들 즉 사건과 장벽, 잠금/신호기발, 림계구역을 제공한다. 사건연산들은 생산자-소비자 동기화를 실현하는데 사용된다. 장벽은 장벽동기화에 사용된다. 잠금과 림계구역들은 주로 호상배제형식에서 원자성을 실현하는데 사용된다. 이 마지막 두개의 구문들은 여기서 평가된다.

신호기발과 잠금

신호기발 S는 다만 두개의 원자연산 P(s)와 V(s)에 의해 처리될수 있는 부가 아닌 옹근수이다.

- P(s)연산은 S가 0보다 크게 될 때까지 프로쎄스를 지연시키는데 사용된다. 그다음 S를 1로 감소시킨다.
- V(s)연산은 단순히 S를 1로 증가시킨다.

2진신호기발 S(즉 S는 True 또는 False이다.)는 **잠금으**로도 불리운다. 잠금 S에 대한 P(s), V(s)연산을 흔히 각각 lock(s), unlock(s)로 쓴다. 잠금의 공동사용은 다음의 실례에서 서술되는바와 같이 호상배제를 통하여 critical section을 원자연산으로 전환하는것이다.

실례 7.3. lock와 unlock연산자의 사용

모든 트랜잭션에서 단일한 잠금 S를 사용한다. 프로쎄스는 임의의 회수와 예금, 구좌이동을 하기전에 먼저 lock(S)를 실행하여 잠금 S를 획득하여야 한다. 그리고 트랜잭션을 끝낸 다음 unlock(s)를 실행하여 잠금을 해제하여야 한다.

프로쎄스가 잠금을 획득하였지만 그것이 아직 해제되지 않았다면 프로쎄스는 잠금을 가지고 있다. 하나의 구좌이동트랜잭션은 아래의 코드로 실현할수 있다.

```
// Account Transfer__Version1
look(s);                          // 입력자료코드
if( balance[x]>100 ) {            // 림계구역
        balance[x]=balance[x]−100;
        balance[z]=balance[z]+100;
   }
unlock(s);                        //코드를 탈퇴한다.
```

잠금문제

잠금의 기본우점은 이미 대부분의 다중처리기들에서 지원한다는데 있다. 7.2.4에서 론의한바와 같이 효과적인 잠금에 대한 많은 연구가 진행되였다. 잠금은 거의 임의의 동기화를 실현하는데 사용할수 있는 대단히 융통성 있는 기구이다. 그러나 호상배제적인 잠금기술들은 원자성을 실현하는데 사용할 때 일부 치명적인 결함을 가지며 다음의 문제들을 일으킬수 있다.

(1) **비구조성** lock들은 구조화된 구문이 아니므로 그것들의 사용은 오유를 일으킬수 있다. 콤파일러는 틀린 코드가 여전히 합법적이므로 lock/unlock구문이 틀리거나 파잉이면 검사할수 없다. 비렬거가능성 및 혼잡문제들에 관한 코드작성은 아래에서 론의하는바와 같이 쉽다.

(2) **파잉서술** 잠금은 실제로 사용자가 원하는것이 아니다. 사용자는 트랜잭션이 자동적으로 진행되는데만 관심을 가진다. 잠금은 원자성을 성취하는 유일한 방법이다. 이 파잉서술은 7.2.3에서 론의하는 보다 효률적인 잠금해제기술들을 가능한 배제한다. 하나의 특정한 실현에 대한 파잉서술은 이식성도 파괴하며 코드를 리해하기 힘들게 만들수 있다.

(3) **상태의존성** 트랜잭션은 바로 무조건적인 원자연산이다. 그러나 잠금해답은 신호기발 S를 도입하여 조건적인 원자연산 lock(S)를 사용할것을 요구한다. 프로쎄스는 lock(s)를 통과하거나 신호기발변수 S의 값에 의존하지 않을수 있다. 이러한 자료의 상태의존동기화는 상태독립동기화보다 일반적으로 리해하기 더 힘들다. 이것이 실제로 간단한 구좌이동코드의 문제는 아니지만 잠금이 문제를 일으키는 보다 복잡한 경우들이 많다.

(4) **순차실행** *n*개의 처리기들 매개가 하나의 트랜잭션을 실행하려 한다고 하자. 잠금해제의 호상배제적인 본질에 의해 이 *n*개의 트랜잭션은 그것들이 서로 다른 구좌에 접근하고 있다고 해도 한번에 하나만 실행되여야 한다. 또한 이 순차적인 실행은 사용자가 원하는것이 아니다. 병행성을 높이기 위하여 흔히 제기되는 착상은 매 구좌마다 분할된 신호기발을 사용하는것이다. 그러나 간단히 알수 있는바와 같이 이 착상은 항상 동작하지 않는다.

(5) **잠금부가처리** O(n)의 lock(s)와 unlock(s)연산들을 순차적으로 실행할 때 추가적인 부가처리도 있다. 더우기 *n*개 처리기들 매개가 하나의 lock(s)를 실행할 때 최대로 그것들중 하나가 성공할수 있다. 다른 프로쎄스들은 반복적으로 S에 접근을 시도해야 한다. 이것은 많은 기억기통신흐름을 생성할수 있다. 공유변수 S에 대

한 경쟁은 분쟁문제[499]를 일으켜 성능을 더욱 떨굴수 있다.

(6) **우선권전환** 이 문제는 낮은 우선권의 프로쎄스가 선점되고 더 높은 우선권의 프로쎄스에 필요한 잠금을 가질 때 일어 난다. 후자는 잠금에 의해 차단되므로 일어 날수 없다.

(7) **보호차단** 잠금을 가지는 프로쎄스가 페지오유나 시간초과로 중단될 때 잠금을 기다리는 다른 프로쎄스들은 진행할수 없다[306].

(8) **비렬거성** 실례 7.3에서 구좌이동트랜잭션이 진행되고 있는 동안 다른 프로쎄스는 어떤 구좌에도 접근할수 없다. 직관적으로 더 좋은 해답은 매 구좌마다 분리된 잠금를 사용하는것이다. 착상은 하나의 구좌가 어떤 프로쎄스에 의해 잠그어 질 때 다른 프로쎄스들은 여전히 다른 구좌들에 접근할수 있다는것이다. 다음의 코드는 이 착상에 기초한것이다.

```
//Account  Transfer__Version2
    lock(S[X]);                              //구좌 X(를 위한 기발)를 잠금
    if(  balance[X]>100  )  {
        balance[X]=balance[X]−100;
        unlock(s[X]);              //구좌 X의 리용을 끝낸다
       lock(S[Y]);                 //구좌 Y를 잠근다
       balance[Y]=balance[Y]+100;
       unlock(s[Y]);               //구좌 Y의 리용을 끝낸다
       }
    else
       unlock[S[X]];               //구좌 X가 열리도록 한다
```

else분기를 잊기 쉬운데 이것은 콤파일러가 발견할수 없는 오유를 일으킨다. 구좌 X에 충분한 자금이 없을 때 unlock(s[X])를 포함하는 **if**문의 true분기가 실행되지 않는다. 그리하여 구좌 X를 영원히 잠그어 앞으로 임의의 프로쎄스도 거기에 접근할수 없다.

이 코드는 구좌 X는 항상 열려 져 있다고 담보한후에도 계속 동작하지 않는다. 이때 오유는 그다지 명백치 않다. 문제는 우의 코드가 항상 렬거가능하지 않다는것 즉 만일 여러 트랜잭션이 동시에 실행될 때 그 결과는 이 트랜잭션들이 어떤 임의의 순차로 실행된것과 꼭 같지 않을것이라는것을 의미한다. 비렬거성은 원자성을 위반한다.

(9) **교착** 렬거성을 담보하는 잠금방법들이 있는데 그것들중 하나는 2단잠금규약으로 알려 져 있다. 이것은 원래 자료기지체계들에서 동시조종을 위하여 개발되였다. 이 방법에서 모든 lock연산들은 트랜잭션의 첫번째 unlock전에 실행되여야 한다. 아래에 제시한것은 2단계잠금을 사용하는 코드이다.

```
//Account  Transfer__Version3
    lock(S[X]);                              //구좌 X(를 위한 신호기발)를 잠금
```

```
    if(  balance[X]>100  ) {
          balance[X]=balance[X]−100;
          lock(S[Y]);                     //구좌 Y(를 위한 신호기발)를 잠금
          balance[Y]=balance[Y]+100;
          unlock(s[Y]);                   //구좌 Y리용을 끝낸다
    }
unlock[S[Y]];                             //구좌 X의 리용을 끝낸다
```

이 코드는 렬거가능이며 Version2보다 약간 더 단순하다. 그러나 그것은 동시성을 제한한다. 한개의 프로쎄스가 구좌 X의 프로쎄스를 끝냈고 구좌 Y를 처리하고 있을 때 구좌 X는 계속 잠그어 져 있다. 그리하여 구좌 X는 이 처리에 더 필요하지 않지만 다른 처리들은 그것을 사용할수 없다.

물론 Version3은 임의의 동시성도 허락하지 않는 Version1보다 더 높은 동시성을 허용한다. Version3코드의 더 치명적인 문제는 혼잡의 가능성이다.

처리 P가 구좌 X에서 구좌 Y로 100$를 이동하고 있는 동안 다른 처리 Q는 구좌 Y로부터 구좌 X에로 50$를 이동하고 있다고 하자.

*P*와 *Q*가 병렬로 실행되고 있기때문에 비승리정황에 빠질수 있다. 즉 P가 X에 대한 잠금으로 Y에 대한 잠금을 획득하려고 시도하는 동안 Q는 Y에 대한 잠금으로 X에 대한 잠금을 획득하려고 시도하고 있다. 2단잠금규약에 따라 프로쎄스는 모든 잠금들을 얻기전에 잠금을 해제할수 없다. 그리하여 그것들중 아무도 요구하는 잠금을 얻을수 없다. 그것들은 **lock**(S[Y])에서 기다리는 P와 **lock**(S[X])에서 기다리는 Q로 하여 혼잡된다.

이러한 교착들을 처리하는 여러 방법들이 있지만 병렬프로그람작성에 적합하지 못하다.

실례로 교착발견방법을 가지는 실시간지원체계는 lock/unlock연산들의 실행을 조종하거나 기다림그라프를 만들수 있다.

마디는 실행하고 있는 매 트랜잭션마다 창조된다. 트랜잭션 P가 현재의 트랜잭션 Q가 가지고 있는 잠금 S를 기다리기 위한 lock(S)를 실행할 때 가지(호)는 P로부터 Q에로 그려 진다. 교착은 기다림그라프가 순환을 가질 때 일어 난다.

일단 교착이 발견되면 순환의 어떤 트랜잭션은 중단될수 있으며 프로그람의 상태는 트랜잭션의 효과를 없애도록 복구된다. 이 방법은 트랜잭션부하가 작을 때 자료기지의 동시성조종에 효과적이다. 그러나 병렬계산프로그람에서 기다림그라프를 창조하는 실시간부가처리와 중단과 복구의 실시간부가처리는 너무 큰것 같다.

다른 방법은 교착예방이다. 즉 프로그람은 프로쎄스들이 절대로 고리형의 기다림상태에 들어 가지 않도록 작성된다. 이것을 달성하는 한가지 방법은 모든 잠금들을 순서 짓고 (실례로 신호기발이름들의 사전식순서를 사용한다.) 여러개의 잠금을 요구하는 임의의 프로쎄스는 항상 그 순서로 그것들을 획득하게 담보하는것이다. 그러나 이 계획은 동시성을 제한하며 프로그람작성자에게 추가적인 부담을 준다. 결과 코드는 너무 복잡해 진다.

이러한 문제들을 극복하기 위하여 아래에서 론의되는 여러개의 구문들이 유도되

였다.

표 7-2에서는 이 구문들을 사용자의 관점에서 비교한다.

사용자에 대하여 가장 좋은 원자성구문은 원자구역이며 가장 좋은 호상배제구문은 림계구역이다.

표 7-2 다변수트랜잭션들을 위한 구조들의 비교

구조문제	신호기발 또는 잠금	림계구역	검사와 설정	비교와 교환	트랜잭션기억기	꺼내기와 첨가
비구조화	xxx		xxx	Xxx	xxx	xxx
파잉서술	xxx	xx	xxx	Xxx	xxx	xxx
상태의존성	xxx		xxx	Xxx	xxx	xxx
순차실행	xxx	xxx	xxx	X	xxx	
부가처리	xxx	xxx	xx	X	x	
우선권전환	xxx	xxx	xxx	X	x	
보호차단	xxx	xxx	xxx	X	x	
비렬거가능	xx		xx	Xx	x	xxx
교착	xx		xx	xx	x	xxx

Xxx:불허 또는 극히 극복하기 힘듬
Xx: 약한 불허 또는 사용자가 주의깊게 코드를 작성하면 완화될수 있음
X: 약함 또는 사용자가 어떤 잘 정의된 규칙에 따르면 없앨수 있음
빈 칸: 문제가 없음, 체계에 의해 수행됨.

림계구역

critical section의 호상배제를 담보하는 구조화된 구문이 critical region인데 다음과 같은 문법을 가진다.

```
critical_region    resource
{                                    // 입력자료점
    S1;S2;…;Sn;                      // 림계구역
}                                    // 탈퇴점
```

여기서 resource는 공유된 변수들의 모임을 표시한다. 착상은 같은 자원을 공유하는 모든 critical region들은 호상배제적으로 실행되여야 한다는것이다. 이 요구는 체계(콤파일러와 실시간지원)에 의해 실시된다. 이 구문은 원래 조작체계응용을 위하여 제안되였다. 병렬프로그람작성에 사용될 때 두개의 수정이 가해 진다.

첫째로, 자원부분은 실제로 쓸모 없으며 그래서 빠진다. 실지 다중처리기들에서 사용한 림계구역구문은 다음의 문법을 가진다.

```
critical_region            //  잠금과 동등한 코드
{                          //  lock(s)
S1;S2;⋯;Sn;                //  S1;S2;⋯;Sn;
}                          //  unlock(s)
```

둘째로, 그러한 다중처리기들에서 림계구역은 우에서 보는바와 같이 구조화된 잠금 방법임을 의미한다. 체계는 정확한 신호기발 S를 자동적으로 선언하고 초기화하여 정확한 lock/unlock문들을 생성한다.

림계구역은 잠금에 비해 많은 우월성을 가진다. 그것은 구조화되고 상태독립이므로 사용하기 더 쉽다. 림계구역을 사용하는 트랜잭션들은 렬거가능하며 교착이 없다. 그것은 잠금에 비해 더 작은 파잉서술이며 구조에 적게 의존한다. 림계구역은 바로 호상배제적으로 실행될수 있는 코드의 쪼각이라고 할수 있다. 이것은 잠금이 사용되지 말아야 한다는것을 의미하는것은 아니다. 현재 잠금은 호상배제를 실현하는 가장 대중적인 기술이며 더 좋은 다른 방법들도 있다.

7. 2. 3. 저수준동기화원시지령

많은 다중처리기들의 하드웨어는 요소변수에 대한 개별적인 읽기 또는 쓰기연산들의 원자성을 담보한다. 또한 대부분의 다중처리기의 하드웨어는 일부 원자성명령들을 제공하는데 그 매개는 요소변수에 대한 단일한 read-modify-write연산을 실현한다. 그러한 명령들은 더 크게 확장된 원자연산들을 실현하기 위하여 함께 사용될수 있다. 아래에서는 복잡성이 증가하는 순서로 4개의 저수준구문들을 론의하고 잠금을 실현하는데 어떻게 리용되는가를 본다.

검사와 설정

TAS(S, temp)로 표현되는 test_and_set명령은 공유변수 S의 값을 국부변수 temp에로 읽고 S를 true로 설정하는 원자연산이다. test_and_set의 주요쓰임은 아래에서 보여 주는바와 같이 잠금을 실현하는것이다.

```
while(S)                           //  이 세개의 행은
TAS(S,temp)                        //  하나의 lock(s)연산을
while(temp)TAS(S,temp);            // 실현한다
if(balance[x]>100){                // 림계구역
balance[x]=balance[x]-100;
        balance[z]=balance[z]+100;
}
S=False                            //  unlock(s)
```

매 TAS(S,temp)의 실행은 공유변수 S에 쓸것을 요구한다. 이것은 무거운 기억기접근 통신흐름을 일으킬수 있다. 우의 **lock**(s)실현은 연산 test-test-and-set를 사용한다. 첫번째

while순환은 국부캐쉬에서 S의 복사를 반복하여 검사한다. 프로쎄스는 잠금 S가 다른 프로쎄스에 의해 해제되였음을 발견할 때만 첫번째 **while**순환을 떠난다.

비교와 교체

compare_and_swap명령 CAS(S, Old, New, Flag)는 공유변수 S의 값을 국부변수 Old의 값과 비교하는 원자연산이다. 이 두개 변수의 값이 같으면 국부변수 New의 값은 S에 기억되며 국부 Flag는 S가 수정되였음을 가리키는 True로 설정된다. 두 값이 같지 않으면 변수 Old는 S의 값을 가지며 기발은 False로 설정된다. compare_and_swap의 의도적인 사용은 다음의 구좌예약코드에서 보여 준다.

```
    Old=balance[X];                        //  공유변수읽기
    do {
    New=Old−100;                           //  수정
    CAS(balance[X],Old,New,Flag);          //  쓰기
  } while (Flag = = False);
```

대비적으로 같은 구좌예약연산은 잠금에 의해 실현될수 있다.

```
    lock(s);
    balance[X]=balance[X]−100;             //  read_modify_write
    unlock(X)
```

잠금실현은 전체(읽기와 수정, 쓰기) 렬을 호상배제로 만든다. 그러나 compare_and_swap실행은 여러 처리기들이 동시에 읽기와 쓰기를 실행하도록 한다. 쓰기만은 호상배제연산(CAS지령)으로 진행된다. 그러므로 compare_and_swap사용의 우월성은 critical section의 길이를 바로 한개의 지령으로 줄일수 있다는것이다.

compare_and_swap는 여러가지 부족점들도 가진다.

첫째로, 그것이 저수준구문이며 사용하기 힘들다는것이다.

둘째로, 하나이상의 공유변수에 접근하는 트랜잭션을 실현하는데 compare_and_swap를 사용할수는 있지만 힘들다는것이다. 이것은 구좌예약실례를 왜 이 부분에서 사용하는가 하는 리유로 된다. 구좌이동트랜잭션을 실현하는 compare_and_swap코드는 훨씬 더 복잡하다.

세번째 부족점은 다음의 ABA문제이다. CAS(S, Old, New, Flag)명령에서 절대적인 가정은 Old=S일 때 S는 수정되지 않았다는것이다. 그러나 이 가정은 항상 성립하지 않는다. 실례로 프로쎄스 P가 구좌 [X]에서 A딸라의 값을 읽는다고 하자. 그다음 프로쎄스 *Q*는 200$를 X에 예금하여 balance를 B=A+200으로 변화시킨다. 그다음 프로쎄스 R는 X로부터 200$를 꺼내서 balance를 A로 되돌려 보낸다. 그래서 그후 프로쎄스 P가 compare_and_swap를 실행할 때 구좌 X는 여전히 A딸라를 가지며 구좌는 변경되지 않았다고 결론할것이다. 그러나 실제로 그것은 수정되였다.

이 ABA문제는 간단한 구좌예금코드의 정확성에는 영향을 주지 않는다. 그러나 다른 경우 특히 공유변수가 지시자일 때 부정확한 결과를 초래할수 있다.

ABA문제에 대한 해답은 [349]에서 제안되였는데 후에 많은 현대 RISC처리기들에서 실현되였다. 이 기술은 2개의 원자명령을 제공한다. 프로쎄스 P는 공유변수 S의 값은 국부등록기 Local에 그리고 S의 주소는 국부보존등록기에로 읽기 위하여 read_and_reserve명령 RAR(Local, S)를 실행할수 있다. 이 명령은 또한 보존등록기와 관련된 1 bit의 꼬리를 0으로 설정하여 S가 다른 처리기들에 의해 수정되지 않았음을 가리킨다. 다음 프로쎄스 P는 S에 대한 새로운 값 New를 국부적으로 계산할수 있다. 그후 P는 write_if_reseved명령 WIR(S, New, Flag)를 실행하여 S를 갱신한다. 이 지령은 먼저 꼬리가 0인가를 검사한다. 만일 0이면 S는 New의 값으로 성과적으로 갱신된다. 그렇지 않으면 갱신은 실패한다. 갱신이 성공하면 기발은 1로 설정되며 그렇지 않으면 0으로 설정된다.

만일 P가 wirte_if_reserved지령을 실행하기전에 S가 다른 프로쎄스에 의해 수정되면 보존등록기의 꼬리는 하드웨어에 의해 자동적으로 1로 설정된다.

트랜잭션기억기와 Oklahoma갱신

방금 론의한 보존기구는 단일공유변수의 원자적갱신에 적합하다. 최근의 결과는 다중보존방법으로 여러개의 공유변수들에 대한 잠금해제의 원자적갱신을 가능케 한다.

이 방법은 DEC[307]의 Herlihy와 Moss 그리고 IBM의 Stone et al.[589]에 의해 독립적으로 개발되였다. DEC판본은 Transactional Memory라고 부르며 IBM판본을 **Oklahoma**갱신이라고 부른다.

여기서는 실행에 상세히 들어 가지 않고 프로그람작성자의 견지에서 기본적인 사상을 론의한다. 두 그룹은 다중보존이 현재의 캐쉬일관성하드웨어의 우월성을 가지는 다중처리기에서 효과적으로 실현될수 있다는것을 보여 주었다. DEC의 론문은 보다 프로그람작성자지향이므로 다중보존방법을 참조할 때는 술어 트랜잭션기억기를 사용한다. 트랜잭션기억기는 다음의 원자명령들을 가지는 트랜잭션의 효과적인 잠금해제의 실현을 제공하는것이다.

- 우에서 론의한 read_and_reserve지령 RAR(Local, S).
- tentative_write지령 TW(S, Local). 이것은 Local의 값을 림시적으로 공유변수 S에 쓴다. 그러나 새로운 값은 전체 트랜잭션이 성과적으로 담보될 때까지 다른 프로쎄스들에서 보이지 않는다.
- 담보명령 COMMIT. 이 명령은 다른 프로쎄스들이 이 트랜잭션에서 접근한 공유변수들을 갱신하지 않았다면 성공하며 True를 되돌린다. 그렇지 않으면 그것은 실패하며 False를 되돌린다. 만일 COMMIT가 성공하면 그것은 즉시에 모든 림시적인 쓰기를 다른 프로쎄스들에 보이게 한다. 실패하면 모든 림시적인 쓰기는 버려 진다.

이 지령들을 사용하여 구좌이동트랜잭션은 그림 7-7에서 보여 준바와 같이 실현될수 있다. 프로그람은 먼저 공유기억의 읽기와 수정, 림시적인 쓰기를 수행한다. 다음 트랜잭

션의 담보(commit)를 시도하며 트랜잭션이 성과적으로 담보되면 while순환을 탈퇴한다. 그렇지 않으면 트랜잭션의 시작에로 되돌아 와 다시 시도한다. 주석 붙은 행들은 기억기 충돌을 줄이기 위한 지수적 backoff라고 하는 잘 알려 진 기술을 실현한다. 그 착상은 일단 트랜잭션이 중단(즉 담보에서 실패)되면 그것의 다음번 시도까지 일정한 시간동안 지연될것이라는것이다.

```
while ( True ) {                          //  이 고리는 트랜잭션을 표시한다.
        RAR(oldX, balance[X]) ;           //  공유변수를 국부 oldX로 읽어 넣는다.
        if ( oldX <= 100 ) break ;        //  자금이 불충분하면 트랜잭션을 끝낸다.
            newX = oldX - 100 ;           //  국부 newX의 값을 계산한다.
            TW(balance[X], newX) ;        //  구좌 X에 림시로 쓴다.
            RAR(oldY, balance[Y]) ;       //  공유변수를 국부 oldY로 읽어 넣는다.
            newY = oldY + 100 ;           //  국부 newY의 값을 계산한다.
            TW(balance[Y], newY);         //  구좌 Y에 림시로 쓴다.
            if ( COMMIT() ) break ;       //  충분히 담보되면  끝낸다.
//          WaitTime=random()%(01<<backoff)           // 그렇지 않으면
//          while ( WaitTime -- ) ;                   // 지수철충을 하고
//          if (backoff<BACKOFFMAX) { backoff++; }
}                                                     // 고리로 되돌아 온다.
```

그림 7-7. 트랜잭션기억을 사용하는 구좌이동트랜잭션

우에서 론의한 기술들

꺼내기와 첨가(fetch_and_add)

우에서 론의한 기술들(잠금, 림계구역, test_and_set, compare _ and_swap, 트랜잭션기억기)은 원자성을 순차적으로 실현한다. *n*개 트랜잭션들이 실행될 때 그것들은 한번에 하나만 실행되여야 한다. 이것은 compare_and_swap나 트랜잭션기억기와 같은 잠금해제방법들에서도 성립한다. 왜냐하면 *n*개의 프로쎄스모두가 *n*개의 트랜잭션모두를 병렬로 실행할수 있다 해도 오직 하나의 트랜잭션만이 성과적으로 담보(commit)할수 있으며 다른 *n*−1개의 프로쎄스들을 다시 시도하여야 하기때문이다. 그러므로 *n*개의 트랜잭션을 실행하는데 *O*(n)시간이 요구된다.

fetch_and_add명령 Result=F&A(s, v)는 공유변수 S의 값을 국부변수 Result로 되돌리고 국부값 V를 S에 더하는 원자연산이다. fetch_and_add지령은 여러 트랜잭션들의 병렬실행을 실제로 가능하게 하는 수단이다. fetch_and_add를 사용하는 구좌예금코드는 바로 한개의 명령이다.

F&A(balance[X],100);

이 코드는 단순할뿐아니라 이전의 코드보다 더 빠르다. Combining(결합)이라고 하는 기술에 의해 *n*개 처리기들이 O(logn)시간에 *n*개의 fetch_and_add명령(이것들은 같은 공유변수에 접근한다.)을 동시에 실행하는것이 가능하다[380].

결합기술은 NYU Ultracomputer[276]와 IBM RP3[500]에서 하드웨어적으로 실현되였다. 쏘프트웨어적으로도 실현가능한데 이것은 쏘프트웨어결합(software combining)[272]으로 알려 져 있다.

fetch_and_Φ

Kruskal et al.[380]은 결합(combining)이 fetch_and_add에 속박되지 않는 일반적인 기술이라는것을 보여 주었다. 그들은 read_modify_write연산들과 이 연산들의 결합에 대한 일반적인 형식화를 제기하였다. 사실 많은 fetch_and_Φ연산들은 결합될수 있다. 여기서 Φ는 임의의 불(Boolean) 또는 산수연산들과 같은 간단한 연산들로 된 하나의 큰 모임이다. 만일 같은 공유변수에 접근하는 *n*개의 fetch_and_Φ연산이 동시에 실행되면 그것들은 결합될수 있으며 완료하는데 O(log2n)시간밖에 걸리지 않는다는것이 밝혀 졌다. 원자성을 지원하기 위한 fetch_and_Φ사용에는 기본적으로 3가지 부족점이 있다. 이 문제점들을 아래에 서술한다.

- 첫번째는 fetch_and_Φ연산들이 비싼 결합망에 대한 요구로 하여 실제로 현재의 병렬콤퓨터들의 잠금과 같이 널리 지원되지는 않는다는것이다. 이 부족점은 최근의 발전에 의해 완화되였는데 이것은 결합망이 효과적으로 값 눅게 실현될수 있으며[204] 결합(combining)은 쏘프트웨어적으로 실현될수 있다는것을 보여 준다[272].
- 두번째 부족점은 fetch_and_Φ가 다만 단일한 공유변수를 원자적으로 갱신할수 있을뿐이라는것이다. 실례로 fetch_and_Φ를 리용하여 구좌이동트랜잭션을 실현하는것은 더 힘들다.
- 세번째 부족점은 보다 치명적인데 fetch_and_Φ는 저수준의 민감한 구조이고 test_and_set나 compare_and_swap, 트랜잭션기억보다 더 복잡하며 따라서 일반프로그람작성자가 정확히 사용하기 더 힘들다는것이다.

7. 2. 4. 고속잠금기구

두가지 형태의 동기화연산 즉 잠금과 장벽이 현재의 공유기억기다중처리기들에서 이미 지배적이다. 이 책에서는 장벽이 조종동기화구성적인것으로 하여 훨씬 더 단순한 의미를 가지기때문에 잠금에 초점을 둔다. 효률적인 또는 하드웨어화된 많은 동기화의 실현이 존재한다.

실례들은 Johnson et al.[351], Scott[540], Shang과 Hwang[552]에서 찾아 볼수 있다. 또한 현재의 병렬프로그람작성언어의 다른 동기화구성들도 흔히 잠금에 의해 실현된다.

최근에 기억기나 망통신흐름을 줄이고 분쟁문제를 완화시켜 잠금의 부가프로쎄스를 낮추는데 목적을 둔 효과적인 잠금에서는 많은 전진이 이룩되였다. 이 잠금기구들은 흔히 호상배제문제를 푸는데 사용된다. 이로부터 이것들을 **고속호상배제알고리듬**이라고도 부른다. 현재의 상업체계들은 거의다 하드웨어알고리듬을 사용한다.

여기서는 기본적인 착상을 설명하기 위하여 두개의 하드웨어알고리듬에 대하여 론의

한다.

호상배제에 관한 최근의 좋은 론문은 Zhang at el.[655]인데 여기서 저자들은 여러개의 제안된 하드웨어 및 쏘프트웨어호상배제알고리듬들의 성능을 비교하고 있다. Kontothanassis et al.[374]는 여러가지 하드웨어동기화알고리듬들을 다중프로그람작성의 견지에서 실례로 여러 프로쎄스들이 한 처리기우에서 실행되도록 하는 견지에서 평가하였다.

입장권알고리듬(Ticket Algorithm)

기본착상은 Zhong et al.[665]로부터 채택된 다음의 코드에서 보여 주는바와 같이 매개 프로쎄스가 먼저 입장권을 가진 다음 critical section실행을 위하여 자기의 순서를 기다리는것이다.

```
int Ticket=0, Turn=0;                    //  공유변수를 0으로 초기화
parfor( i=0;k<n;i++ )                    //  n개의 프로쎄스들이 있다
{ int Myticket;                          //  국부변수
    while( Condition ){
    independent computation,noncritical section
    Myticket=FAA(Ticket,1); {            //  이 3개의 행은
    while( Myticket!=Turn )              //  잠금연산과
        Delay(Myticket_Turn);            //   같다
         critical section
    Turn=Turn+1;                         //  잠금해제
   }                                     //  호상배제코드의 탈퇴점
    independent computation,noncritical section
  }
}
```

매 프로쎄스 P는 그림 7-8에서와 같이 3개의 변수를 사용한다. 국부변수 Myticket는 캐쉬나 국부기억기에 기억되며 다른 프로쎄스들이 접근할수 없다. 두개의 대역변수 Ticket와 Turn은 모든 프로쎄스들이 접근가능한 공유기억기안에 있다.

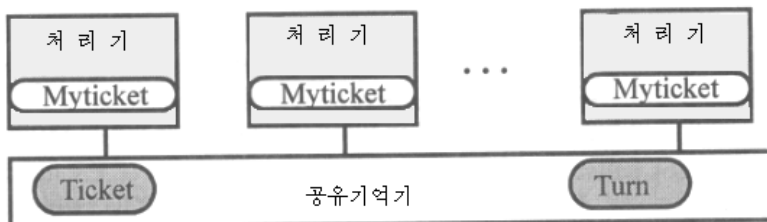



그림 7-8. 입장권알고리듬에서 변수분포

critical section에 들어 가기 위하여 프로쎄스 P는 먼저 Ticket의 수를 국부변수 Myticket안에 넣고 Ticket를 1증가시키기 위하여 원자적인 fetch_and_add 지령을 실행한

다. 그다음 프로세스는 자기 차례가 되기를 기다리면서 회전한다. 공유변수 Turn을 읽는 회수를 줄이기 위하여 프로세스 P는 Turn을 다시 읽기전에 1회기간동안 지연한다. 이 지연시간은 프로세스 P의 앞에서 critical section을 실행하려고 기다리는 프로세스의 수인 Myticket와 Turn의 차에 비례한다. critical section을 실행한후 프로세스 P는 Turn을 증가시킴으로써 잠금을 해제한다.

Turn=Turn+1이 read_modify_write연산이지만 원자적이지 말아야 한다는데 주의하시오. 입장권알고리듬은 원자적fetch_and_add(fetch_and_increment)지령을 제공하는 다중처리기들에 적당한 간단한 회전잠금방법이다. 그것은 FIFO속성을 가진다. 즉 더 작은 표번호를 얻는 프로세스들이 먼저 critical section을 실행한다. 이때 두개의 정수변수 Ticket와 Turn이 초과되지 않도록 담보하여야 한다.

배렬련결식알고리듬(Array_link_Based Algorithm)

입장권알고리듬에서 *n*개의 처리기들은 두개의 변수 Ticket와 Turn에 접근하여야 한다. 충돌을 줄이기 위하여 알고리듬은 Ticket에 접근하고 증가시키기 위해 하나의 fetch_and_add를 사용한다. 모든 *n*개 처리기들은 결합기술을 리용하여 $2t_M \times \log n$개 주기동안 입장권(Ticket)을 가질수 있다. 여기서 t_M개 주기공유기억기접근이 요구된다.

```
Acquire_Lock(short NodeID, short Lock)
{       int Pred ;
P:      Lock_q[NodeID]->Successor = Nil ;
Q:      Pred = Fetch_and_Store(Lock, NodeID) ;
R:      if (Pred != –1) {
            Lock_q[NodeID]->Get_Lock = 0 ;
            Lockq[Pred]->Successor = Lock_q[NodeID] ;
            while (Lock_q[NodeID]->Get_Lock != 1) ;
        }
}
Release_Lock(short NodeID, short Lock)
{       int Pred ;
X:      if (Lock_q[NodeID]->Successor != Nil)
            Lock_q[NodeID]->Get_Lock = 1 ;
        else {
Y:          Pred = Fetch_and_Store(Lock, –1) ;
Z:          if (NodeID != Pred ) {
                Pred = Fetch_and_Store(Lock, Pred) ;
                while (Lock_q[NodeID]->Successor == 0) ;
                Lock_q[NodeID]->Successor->Get_Lock = 0 ;
            }
        }
}
```

그림 7-9. 배렬련결식잠금지령들

매 처리기는 Turn을 국부등록기 또는 캐쉬에로 복사할수 있으며 Turn이 잠금을 해제하는 다른 처리기에 의해 수정될 때까지 국부복사우에서 돈다. 경쟁을 더 줄이기 위하여 Zhang et al.[665]이 제기한 다음의 배렬련결식알고리듬에서와 같이 잠금이 분산될수도 있다. 이것은 Mellor_Crummey와 Scott[441]이 개발한 알고리듬으로부터 개작되였다. 배렬련결식알고리듬은 다음의 자료구조들을 사용한다. 공유변수 Lock는 값 −1로 초기화된다.

```
typedef struct MLINK{
        struct MLINK*Successor; short Get_Lock;
} Mlink;
short Lock,NodeID;
Mlink*Lock_q[Num_Proc_1];
int Node Count;
```

처리기의 NodeID는 Lock를 획득하고 해방하기 위하여 2개의 절차 **Acquire_Lock**(NodeID, Lock)와 **Release_Lock**(NodeID, Lock)를 사용한다. 이 절차들의 상세는 그림 7-9에 제시한다. 이 절차들을 어떻게 사용하는가는 문제 7.5에 남긴다.

7. 3. TCP/IP 통신규약

망상에서의 통신은 보통 규약층들의 탄창으로 실현된다. 규약은 자료형식과 자료를 망우로 어떻게 전송하는가를 지배하는 규칙들의 모임이다.

규약탄창은 규약조로도 알려 져 있다. 이 부분에서는 TCP/IP조라고 하는 표준적인 통신규약을 론의한다. 비표준적이지만 더 효률적인 규약들은 7.4에서 론의한다.

먼저 7.3.1에서 TCP/IP조의 기본특징을 요약한다. 그다음 7.3.2에서 가장 자주 리용하는 3개의 규약들(UDP, TCP, IP)과 그것들이 이써네트우에서 어떻게 실현되는가를 론의한다.

임의의 프로그람이나 응용들에서 이것들을 실현하기 위하여 프로그람작성자는 응용프로그람대면부를 요구한다.

7.3.3에서 UDP/IP나 TCP/IP리용에 사용할수 있는 소케트라고 하는 표준 API를 서술한다.

7. 3. 1. TCP/IP조의 특징

널리 알려 진 2개의 규약탄창을 그림 7-10에서 보여 준다. 7층열린체계호상접속(OSI)탄창은 국제규격화기구(ISO)에서 개발하였다. 이 규격은 널리 실현되지는 않았지만 통신구조의 개념들과 전문용어를 명백히 하는데 위력하다.

보통 TCP/IP조로 알려 지고 인터네트활동위원회(Internet Activities Board, IAB)가 개발한 인터네트통신탄창은 널리 사용되고 있으며 사실상 표준으로 되였다.

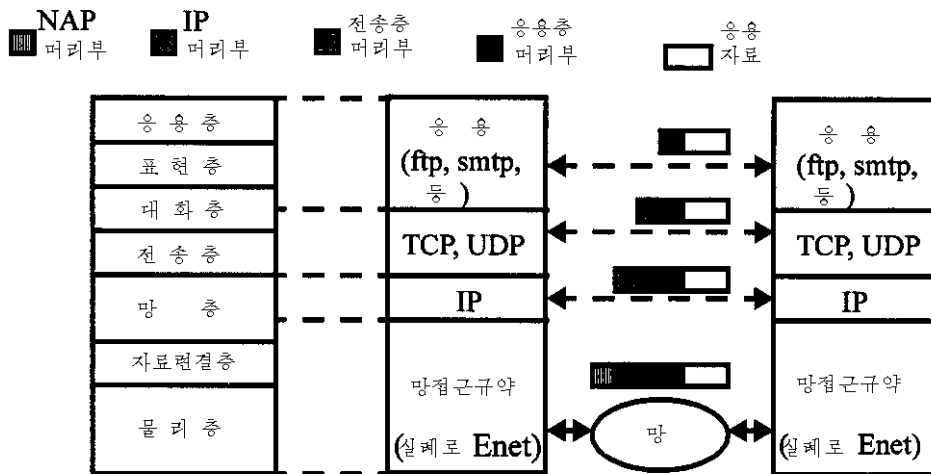



그림 7-10. OSI와 인터네트통신규약탄창들

망안의 두개의 콤퓨터(흔히 호스트라고 부른다.)사이에 통보문을 넘기기하는것은 복잡한 동작이다. TCP/IP조는 이러한 복잡성을 극복하기 위하여 계층화된 구조를 사용한다. 그것은 모든 통신규약조들에 공통인 여러개의 특징들을 가진다.

- **층** 규약층들의 탄창은 통보문넘기기에 사용된다. 매 규약층은 특수한 기능들의 모임을 수행하는데 제공되며 직접 닿은 아래웃층과 직접 호상작용한다. TCP/IP조자체는 3개의 층 즉 응용층(FTP와 TELNBT, SMTP, SUMP와 같은 규약들을 포함)과 전송층(TCP와 UDP규약을 포함), 망층(IP를 포함)을 가진다. 조는 통신하드웨어에 의존하는 망접근층들에서 동작한다.
- **자료교갑화** 응용의 프로쎄스가 자료쪼각을 망으로 전송하려면 통보문을 형성하기 위한 어떤 특별한 정보를 자료에 부가하여야 한다. 이 특별한 정보는 자료의 뒤에 부가될수도 있지만 보통 머리부(header)라고 부른다. TCP/IP조와 같은 규약탄창에서 아래층은 더 높은 층의 전체 통보문을 자기의 자료로 취하고 머리부를 부가하여 자기의 통보문을 구성한다. 더 낮은 층은 더 높은 층의 통보문형식에 관심을 가지거나 리해하지 않는다. 이것은 그림 7-11에서 설명하는바와 같이 **자료교갑화**라고 한다.
- TCP층은 응용통보문에 9개 마당의 TCP머리부를 부가하여 TCP토막이라고 부르는 전송층통보문(통보문이 크면 여러개의 토막)을 구성한다.
- IP층은 TCP토막에 11개 마당의 IP머리부를 부가하여 IP자료문이라고 하는 망층통보문을 구성한다. NAP층은 IP자료문에 6개 마당의 이써네트머리부를 부가하여 이써네트프레임이라고 부르는 NAP층통보문을 형성한다. 인터네트프레임에서 CRC마당은 자료의 뒤에 온다는데 주의해야 한다.
- **동등계층** 한 호스트안의 매개 층은 다른 층들에 상관없이 다른 호스트의 같은 층과 대화한다. 실례로 한 호스트는 이써네트에 접속되고 다른 호스트는 FDDI에 접속되여도 IP층은 IP자료문들이 같은 규약에 따라 전송된다는것을 알고 있다.

- **열린 대면부** 매층은 웃층에 열린 대면부를 제공한다. 가장 중요하게 응용층은 TCP/IP봉사들에 접근하기 위하여 소케트나 TLI(Transport Level Interface)를 사용할수 있다. TCP와 UDP, IP는 보통 핵심부에서 실현되므로 이 대면부들은 체계호출들의 모임으로 나타난다.

응용규약

응용층규약은 규약조가 실현한 기능들에 대한 사용자대면부를 제공한다. 이것은 또

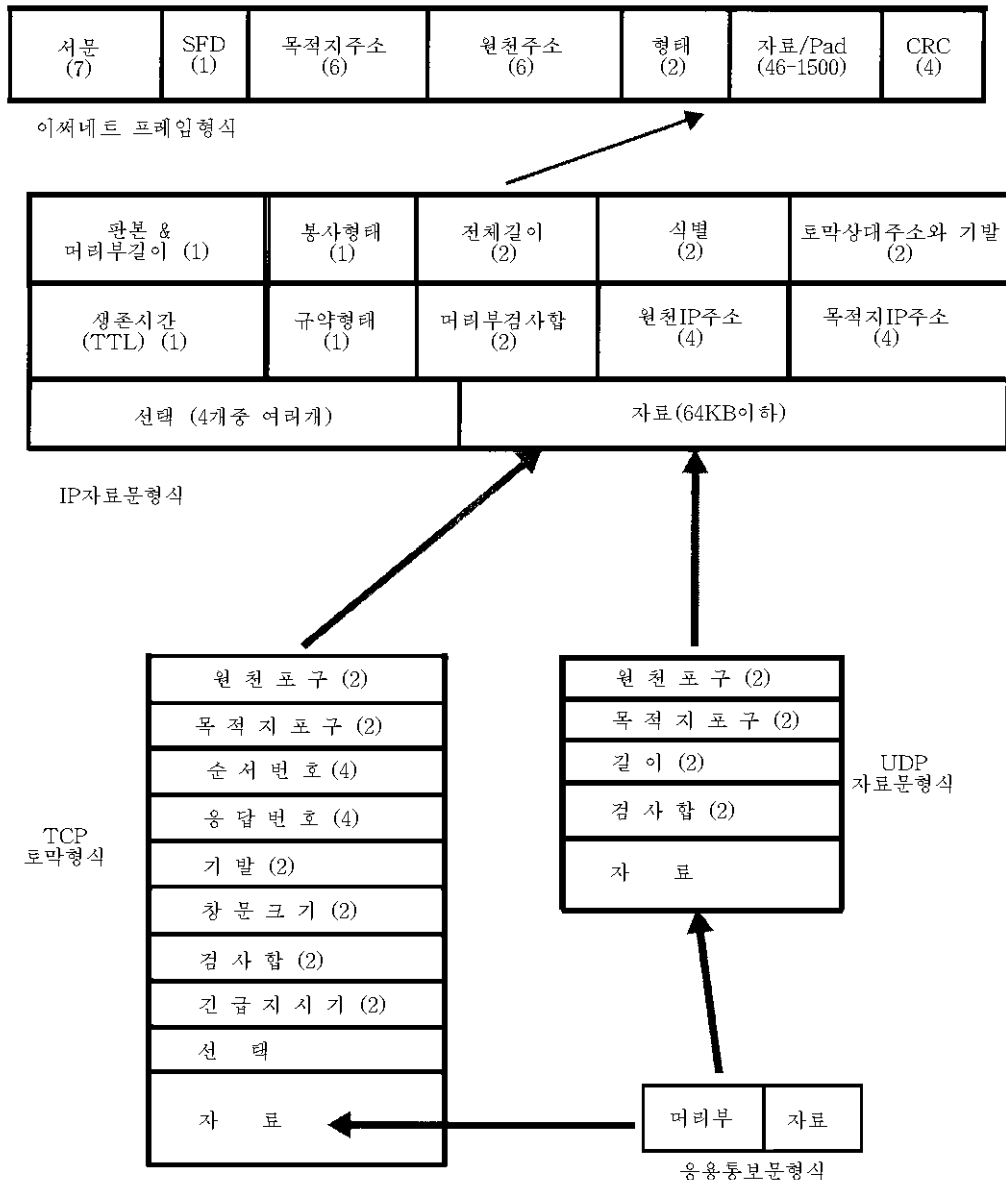



그림 7-11. 이써네트의 TCP/IP에서 여러가지 자료형식

한 원천지의 프로쎄스들이 보낸 통보문은 정확한 목적지의 프로쎄스에 도착한다는 가정하에서 응용특정의 문법적, 의미적문제들로 취급된다.

이 문제들은 작은 끌(little−endian)호스트와 큰 끌(big−endian)호스트가 넘기기 받은 통보문으로부터 같은 론리적자료를 보도록 담보하는것과 트랜잭션프로쎄스체계 등에서 ACID속성들을 가지도록 하는것을 포함한다.

TCP/IP조가 지원하는 많은 응용규약들이 존재한다. 가장 널리 사용되는 응용규약들은 다음과 같다.

- 국부프로쎄스와 원격호스트의 파일체계사이에 파일들을 전송하기 위한 FTP(File Transfer Protocol)
- 원격호스트에로 련결하기 위한 TELNET
- 우편봉사를 위한 SMTP(Simple Mail Transfer Protocol)
- 콤퓨터망의 조종과 감시를 위한 SNMP(Simple Network Management Protocol)
- WWW응용들을 위한 HTTP(Hyper−Text Transfer Protocol)

망주소화

응용층규약은 밑에 있는 규약들에 원격호스트에 있을수도 있는 목적지(와 원천지)프로쎄스의 주소도 알려야 한다. 이것은 아래와 같이 (IP주소, 포구)쌍으로 주어 진다.

159. 226. 43. 150. 23 호스트 IP주소, 포구번호

IP주소는 임의의 목적지호스트를 유일하게 식별하며 포구번호는 목적지의 포구 또는 규약을 가리킨다. 국부Unix호스트의 현존 포구번호들은 《more/etc/servies》를 건반입력하여 찾을수 있다. 포구는 전송층통신의 종단점(끝점)이다. 전송층통신은 항상 한 포구로부터 다른 포구에로 진행된다.

포구번호는 16bit정수로서 호스트당 64k개의 포구들이 있을수 있다. 0부터 1023까지의 포구번호들은 뿌리사용자를 위하여 예약되여 있다. 대부분의 예약된 포구번호들은 잘 알려 져 있으며 누구나 그러한 포구번호들이 규약이나 봉사 그리고 그외의것에 사용되는데 동의한다.

실례로 포구번호 23은 잘 알려 진 포구번호로서 TELNET규약에 사용된다. TELNET규약을 제공하는 체계를 사용하려고 하는 임의의 프로쎄스는 포구번호 23을 사용하여야 한다.

7. 3. 2. UDP, TCP, IP

TCP/IP조는 2개의 전송층규약 즉 TCP와 UDP를 포함한다. 이것들은 다 IP라고 하는 단일한 망층규약우에서 동작한다.

사용자자료문규약(User Datagram Protocol, UDP)

UDP는 대단히 간단한 전송층규약이다. UDP의 단순성이 효률성은 제공하지만 기능

성은 적게 제공한다. 단순성은 그림 7-11에서 보여 주는바와 같이 단순한 자료문형식으로부터 흘러 나온다. UDP머리부는 4개의 16bit마당 즉 원천포구주소, 목적지포구주소, UDP자료문의 총 바이트길이, UDP자료문의 검사합만을 포함한다. 목적지포구와 자료문길이는 항상 요구되지만 나머지 두개는 선택적이다.

원천지포구는 응답이 기대될 때만 필요하다. 원천지포구와 검사합마당이 사용되지 않으면 0으로 채워 진다. 원천지호스트가 목적지호스트로 UDP를 사용하여 일부자료를 보낼 때 먼저 그림 7-11과 같이 UDP자료문을 구성한다. 그다음 두 호스트의 UDP쏘프트웨어는 협력하여 자료문을 목적지호스트에로 넘기기하며 그 다음자료를 추출한다. UDP자료문은 길이를 제한한다는것을 명심하여야 한다. 넘기기되여야 할 자료가 크면 송신자는 그것을 더 작은 토막들로 나누고 분할된 자료문으로 넘기기하여야 한다.

목적지호스트는 역시 이 자료문들을 분할된 통보문들로 접수하고 그것들을 원래의 자료로 다시 조립하여야 한다. 자료문은 목적지에 순서가 없이 도착하거나 전혀 도착하지 않을수도 있다. 이 속성들은 UDP를 비접속성규약, 가장 효률적인 규약으로 되게 한다.

이것은 우편국에 의한 편지보내기와 같다. UDP는 송신자가 접속을 확립하지 않고 자료를 보내는 비접속성이다. 물론 많은 량의 자료는 개별적인 자료문으로 싼 다음 갈라서 넘기기한다.

TCP에서 송신자는 먼저 접속이 확립된 다음 자료를 개별적인 자료문으로서가 아니라 련속적인 흐름(stream)으로 보낸다. UDP는 통보문자료문령역을 유지하지만 TCP는 그렇게 하지 않는다.

UDP는 우편을 배달하기 위하여 최선을 다하는 우편국과 같이 담보가 없기때문에 최대의 노력이 드는 규약이다. UDP자료문은 넘기기도중에 오유가 생기거나 단순히 잃어버릴수 있다. 수신측 UDP는 오유를(검사합을 통하여) 검사하여 오유가 생긴 자료문은 버린다.

UDP는 담보된 배달이 중요치 않는 영상/음성통보문전송이나 때때로 파케트를 잃어 버리는것이 성능에 크게 영향을 주지 않을 때 적당하다. 만일 믿음직한 통보문이(하나의 bit도 흘리면 안되는 프로그람이나 자료파일전송) 요구되면 응용층에서 믿음직한 통보문넘기기기능을 실현하거나 TCP로 절환하여야 한다.

전송조종층(Transmission Control Protocol , TCP)

TCP는 UDP보다 더 강력하고 복잡한 전송층규약이다. TCP는 믿음직한 통보문배달을 담보하는 접속지향의 규약이다. 이것은 전화로 통보문을 보내는것과 같다. 두 단체는 먼저 접속을 확립한 다음 통보문을 넘기기한다. 오유발견과 응답, 재전송기구들에 의하여 통보문은 믿음직하게 넘기기되고 정확히 순서 짓도록 담보된다.

TCP는 쌍방향규약이다. 즉 자료를 나르는 토막들은 원천으로부터 목적지로 보내여지며 빈 자료마당을 가진 응답토막들은 반대방향으로 넘기기된다.

TCP머리부(그림 7-11)는 9개의 마당을 포함한다. 원천포구와 목적지포구, 검사창마당들은 UDP머리부의것과 류사하다. 기발들과 선택적인 마당들은 무시한다.

긴급지시기(urgent pointer)마당은 일단 목적지의 TCP가 자료를 접수하면 즉시 목적지

의 프로쎄스에 보내야 하는 토막의 자료바이트의 수를 가리킨다. TCP머리부에는 토막길이마당이 없다. 왜냐하면 토막길이는 TCP가 계산하기때문이다. TCP는 통보문들을 흐름(stream)으로 넘기기한다. 즉 일단 TCP접속이 확립되면 하나 또는 그이상의 통보문들이 접속을 통하여 흐름으로 보내여 진다.

목적지에서 TCP는 이 흐름을 TCP토막들로 나누고 토막들을 통보문으로 재조립하여야 한다. TCP는 또한 순서가 바뀐 토막들과 중복된 토막들, 믿음직한 넘기기, 흐름조종을 처리하여야 한다.

나머지 3개 마당들은 이 목적을 달성하는데 사용된다.

순서번호는 원천 TCP에 의해 채워 지며 흐름의 기초주소로부터 토막의 첫번째 바이트의 바이트상대주소를 가리킨다.

이것은 목적지 TCP가 순서가 바뀌거나 중복된 토막들을 관리하는데 사용된다. 응답번호는 토막이 수신된다는것을 가리키기 위하여 목적지 TCP에 의해 채워 진다.

창문크기도 목적지 TCP에 의해 채워 지는데 목적지 TCP가 받기 위하여 준비되여 있는 현재의 응답된 토막이외의 자료의 량을 가리킨다. 이 값은 흐름조종에 사용되며 이 접속에 리용가능한 목적지 TCP의 완충기령역에 의존한다. TCP는 이 정보를 다음번에 보내야 할 토막의 길이로 선택하는데 사용한다.

길이선택알고리듬은 높은 대역너비를 유지하는 한편 목적지쪽의 혼잡을 조종하려고 노력한다.

TCP는 믿음직한 통보문넘기기를 위한 재전송에 응답을 사용한다. 원천지쪽의 시간기록기는 토막이 보내질 때 활성화된다. 목적지 TCP는 토막을 수신할 때 응답을 돌려보낸다. 시간계수기가 끝나기전에 원천지가 응답을 수신하면 시간계수기는 취소된다. 시간계수기가 끝났을 때 응답이 수신되지 않으면 원천은 같은 토막을 다시 보낸다. 이 방법은 여러 문제점을 가진다.

첫째로, 원천지가 새로운 토막을 보내기전에 응답의 도착을 기다리면 대역너비가 영향을 받는다. 원천지가 응답을 기다리지 않고 련속적으로 토막들을 보낼수 있다고 생각할수 있다. 그러나 이것은 목적지의 완충기를 빨리 포화시키며 여러가지 엄중한 망혼잡을 일으켜 지연을 증가시키고 대역너비를 줄인다. 더우기 재전송은 목적지에 보내는 한개 토막에 대한 여러개의 복사를 일으키며 혼잡을 더욱 악화시킨다.

TCP는 이러한 문제들을 풀기 위하여 미끄럼창문흐름조종기구를 사용한다. 원천지는 련속적으로 토막들을 내보내게 되는데 이것은 창문크기를 웃한계로 가지며 따라서 엄중한 목적지측의 혼잡을 일으키지 않는다. 목적지는 보내온 매개 토막들에 응답하지 않는다.

그림 7-12에서 고정된 토막크기는 10kB이고 초기창문크기는 100kB 또는 10개 토막이라고 가정하자. 초기의 창문은 토막 1～10을 포함한다.

토막 1부터 10까지 보낸후 원천지는 토막 1부터 5까지 수신되였다는것을 알리는 응답을 수신한다. 목적지는 응답과 함께 새로운 창문크기를 8개 토막으로 하자는것을 보낸다.

이제는 창문이 토막 6부터 13까지를 포함하도록 오른쪽으로 미끄러 진다.

다음원천지는 토막 11부터 13까지 보내는데 성공한다. 토막 6부터 10까지는 망에 있다는것 즉 그것들이 전송은 되였지만 아직 응답되지 않았다는데 주의하여야 한다. 그것

들은 목적지로 가는 도중에 있거나 그것들이 도착하였지만 목적지가 응답을 보내지 않았거나 응답이 보내졌지만 원천지에 아직 도착하지 않았을수 있다.

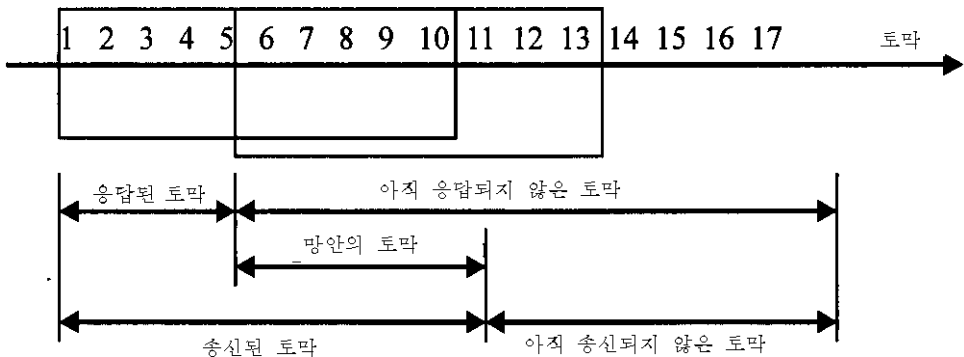



그림 7-12. TCP흐름조종에서의 미끄러 지는 창문기구

인터네트규약(Internet Protocol, IP)

IP의 기본기능은 하나이상의 LAN으로 구성되는 인터네트워크(인터네트)의 한 호스트로부터 다른 호스트로의 통보문들의 경로를 정하는것이다. 즉 통보문은 경로기(관문으로도 알려 져 있다.)라고 하는 하나 또는 그이상의 중간마디들을 통과한다.

인터네트우의 임의의 호스트는 유일한 IP주소를 가진다.

경로기는 적어도 2개의 LAN들에 접속되며 하나이상의 IP주소를 가진다는데서 보통의 호스트와 차이난다. 경로기의 기본기능은 인터네트에서 IP자료문들의 경로를 정하는것이며 한편 호스트는 보통 다른 목적(실례로 탁상 PC, 파일봉사기, 계산봉사기 등)들에 사용된다.

IP는 그것이 비접속성이고 최대의 노력이 드는 규약이라는데서 UDP와 비슷하다. IP자료문들은 순서없이 도착하거나 잃어 질수도 있다. IP는 웃쪽의 층들이 이 문제를 해결하게 한다.

그림 7-13에서 호스트 A가 자료문을 호스트 Z에로 어떻게 보내는가를 서술함으로써 IP기구를 설명한다.

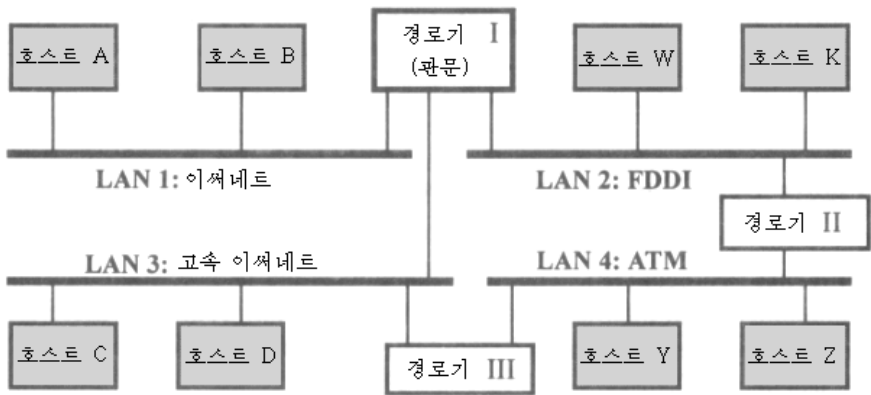



그림 7-13. 3개의 경로기에 의해 련결된 4개의 LAN들을 가진 인터네트

호스트 A의 IP쏘프트웨어모듈은 일단 요청을 받으면 국부적으로 표를 검사하며 호스트 Z가 같은 LAN우에 없다는것을 발견하면 자료문을 경로기 I로 보낸다. 이 경로기는 한번 더 경로를 정하고 자료문을 경로기 Ⅱ나 경로기 Ⅲ으로 전진시키며 그다음 자료문을 최종적인 목적지호스트 Z에 보낸다.

경로정하기기능은 자료문안의 원천 및 목적지IP주소를 사용한다. 경로정하기외에 IP층은 다음의 2가지 기능을 제공한다.

- **분할과 재조립**
  인터네트망의 LAN들은 서로 다른 형태이며 최대파케트크기에 대한 제한도 서로 다르다. 그래서 IP는 큰 자료문을 여러개의 더 작은 파케트들로 분할하고 그것들을 재조립하여야 한다. 이 기능은 IP자료문의 다음의 마당들 즉 전체 길이와 식별자, 토막상대주소와 기발들, 원천/목적지IP주소들을 리용한다.
- **오유통지**
  오유가 있는 경우 오유를 포함한 호스트들은 통지되여야 한다.

봉사형태마당은 앞서기(우선권)와 믿음성, 지연, 처리량에 대한 요구를 서술한다.

생존시간(time_to_live)마당은 자료문의 수명을 주며 죽은 자료문을 파피할 때 리용된다. 그렇지 않으면 자료문은 끝없이 돌것이며 적당치 않는 경로정하기로 망을 방해한다.

규약형태마당은 어느 전송규약(TCP,UDP 등)이 IP를 사용하는가를 서술한다. 식별자마당은 규약형태와 원천/목적지IP주소와 함께 자료문을 유일하게 식별한다.

7. 3. 3. 소케트대면부

소케트는 TCP/IP조를 사용하기 위한 API(즉 자료형태와 기능들의 모임)이다. 소케트는 원래 Berkeley Unix(BSD 4.2)에서 실현되였다. 지금 그것은 Microsoft Windows가동환경은 물론 거의 모든 유닉스체계들에서 실현된 사실상의 표준으로 되였다.

두개의 프로쎄스가 소케트대면부로 통신을 하려면 먼저 그들 매개가 소케트를 창조하고 어느 통신규약을 사용하는가를 서술한다. 그다음 그것들은 각각의 소케트에 대한 읽기/쓰기로써 통신을 한다.

소케트쏘프트웨어는 실제적인 통신의 수행을 담당한다. 여기서는 훨씬 단순화한 령역이름봉사기를 가지고 소케트대면부의 기본사상을 설명한다.

비접속실현(UDP)

UDP실현을 위한 프로그람골조를 그림 7-14에 제시한다. 봉사기와 의뢰기는 모두 처음에 소케트기능에 대한 봉사를 호출한다.

목적은 변수 Mysocket로 지적된 하나의 소케트를 창조하는것이다. 3개의 점들(...)은 인수들을 가리키는데 여기서는 관심을 돌리지 않는다. 상수 AF_INET는 인터네트망의 령역주소(IP주소)가 소케트주소화형식으로 사용된다는것을 가리키며 sock_DGRAM은

```
main ()
{
   Mysocket = socket(AF_INET, SOCK_DGRAM, ...) ;
   bind( Mysocket, ... ) ;
   recvfrom( Mysocket, Hostname, ...) ;
   host_IP = NameToIP(Hostname) ;
   sendto( Mysocket, host_IP, ...) ;
}
```

ㄱ) 봉사기코드골조

```
main (int argc, char *argv[])
{
   Mysocket = socket(AF_INET, SOCK_DGRAM, ...) ;
   sendto(Mysocket, argv[1], ...) ;
   recvfrom(Mysocket, host_IP, ...) ;
}
```

ㄴ) 의뢰기코드골조

그림 7-14. UDP 소케트를 사용하는 단순화된 령역이름봉사기의 코드골조

UDP가 사용된다는것을 가리킨다.

다음 봉사기는 bind함수호출에 의해 소케트를 포구번호에 련결한다(bind). 의뢰기는 명확히 련결하지 않고 련결할 포구번호를 자동적으로 선택할것을 체계에 맡길수 있다.

소케트가 구성된후 의뢰기는 sendto함수를 리용하여 호스트이름을 봉사기로 보낸다. 봉사기는 recvfrom함수로 그 통보문을 접수한 다음 그것을 국부함수 NameToIP를 실행하여 대응하는 IP주소로 변환하며 그 IP주소를 sendto함수에 의해 의뢰기로 돌려 보낸다. 의뢰기는 recvfrom함수에 의해 IP주소를 접수한다.

접속지향실현(TCP)

TCP실현을 위한 코드골조를 그림 7-15에 제시한다. 소케트창조와 련결(binding)은 상수 SOCK_STREAM이 TCP의 사용을 가리키는것을 제외하고 UDP실현과 비슷하다.

UDP와 달리 TCP는 접속지향이다. 봉사기는 listen함수를 실행하여 접속을 받기 위하여 준비되여 있다는것을 핵심부에 알린다. 다음 봉사기는 accept함수를 실행하여 의뢰기로부터 접속요청을 받기 위하여 기다린다. 이 함수는 의뢰기가 connect함수를 실행하여 접속할 때까지 귀환하지 않는다. 확립된 쌍방향접속은 보통의 파일과 꼭같이 동작하며 파일지시기 fP로 지시된다. 이제는 봉사기가 파일접근과 꼭같이 접속에 읽기/쓰기함으로써 의뢰기와 통신할수 있다.

모든 통신이 진행된후에 봉사기는 close(fP)를 실행하여 접속을 닫는다.

```
main ()
{
   Mysocket = socket(AF_INET, SOCK_STREAM, ...) ;
   bind( Mysocket, ... ) ;
   listen( Mysocket, ...) ;
   fp = accept( Mysocket, ...) ;
   read (fp, Hostname, ...) ;
   host_IP = NameToIP(Hostname) ;
   write(fp, host_IP, ...) ;
   close(fp) ;
}
```

ㄱ) 봉사기코드골조

```
main (int argc, char *argv[])
{
   Mysocket = socket(AF_INET, SOCK_STREAM, ...) ;
   connect( Mysocket, ... ) ;
   write (Mysocket, argv[1], ...) ;
   read (Mysocket, host_IP, ...) ;
}
```

ㄴ) 의뢰기코드골조

그림 7-15. TCP 소케트를 사용하는 단순화된 령역이름봉사기의 코드골조

7. 4. 고속효률통신

망의 원대역너비(raw network bandwidth)는 지수적속도로 개선되고 있으며 5년마다 거의 10배씩 증가되고 있다. 기가비트망들은 현재의 고성능콤퓨터(HPC)체계들에서 매우 대중적이다. 테라비트망들은 연구소에서 실현되였다(실례로 후지쯔는 1996년 2월에 1.1 Tb/s의 전송을 달성하였다.).

원대역너비와 사용자가 실제적으로 달성할수 있는것 사이에는 큰 간격이 존재한다.

표 7-3은 여러 HPC체계들의 최대대역너비와 유효대역너비들을 비교한다.

점근선적인 대역너비와 지연은 MPI점대점통신으로 측정된다. 유효대역너비는 128B 통보문을 가지고 공식

$$유효대역너비 = 통보문길이/통신시간 = 128/(t_0+(128/r_\infty)) \quad (7.1)$$

에 의해 평가된다.

표 7-3의 자료는 효률적이라고 생각되는 통신하드웨어와 쏘프트웨어로부터 얻어 진다.

하드웨어의 최대대역너비와 유효대역너비사이의 간격은 상용망들(실례로 이써네트)들과 표준적인 규약들(실례로 소케트)을 사용한다면 더 넓어 진다. 이 간격을 줄이려면 통신성능을 정밀하게 꽉 채우는 요인들을 리해하여야 한다.

7. 4. 1. 통신에서 중요한 문제

많은 요인들이 통신부분체계의 성능에 영향을 준다. 중요한것들을 아래에 렬거한다.

- **통신하드웨어** 마디의 기억기와 I/O구조, 망대면부, 망을 포함한다.
- **통신쏘프트웨어** 쏘프트웨어구조와 알고리듬들을 포함한다.
- **사용자환경** 다중사용자, 다중처리, 다중프로그람작성을 지원하는가, SPMD모형을 사용하는가 등을 포함한다.
- **제공된 통신봉사들** 통보문넘기기와 흐름조종, 고장처리, 보호 등을 포함한다.

표 7-3 4개의 HPC체계의 최대 및 유효대역너비들

파라메터	값	기억기	디스크구동	테프단
용량	단가	64Mb/die	0.2Mb/mm^2	0.1-0.4 Mb.mm^2
	체계	32-128MB	2-4GB	8-16GB
	개선률	2X per 2 yr	25-60% per yr	10X per 10-21 yr
접근시간	현재값	60ns	10-25ms	Seconds
	개선률	3X per 12 yr	21X per 10 yr	N/A
대역너비	현재값	500-1000MB/s	1-5MB/s	0.5-4 MB/s
	개선률	N/A	2X per 5 yr	2X to 6X per 10 yr

통신하드웨어(Communication Hardware)

일반적인 통신하드웨어구조를 그림 7-16에 보여 준다. 성긴결합형구성방식에서 NIC는 마디의 I/O모선(실례로 PCI)에 련결된다. 한 마디에서 다른 마디로 가기 위하여 통보문은 송신마디의 기억기로부터 기억기모선, I/O모선, NIC를 통하여 망에로 간다. 수신측에서는 반대순서로 같은 경로를 지나가야 한다. 달성가능한 통신대역너비는 통신경로의 가장 느린 구성요소에 의해 제한된다.

실례 7.4. 콤퓨터들의 클라스터에서 병목들

클라스터는 800MB/s의 기억기모선과 133MB/s의 I/O모선, 200MB/s의 NIC DMA를 가지는 마디들을 사용한다. 다음의 두가지 물음이 제기되고 있다.

(1) 만일 통신망이 1 Gb/s의 최대속도를 가진다면 통신대역너비상의 하드웨어제한은 무엇인가?
(2) 망이 400MB/s로 높아 지면 어떻게 되겠는가?

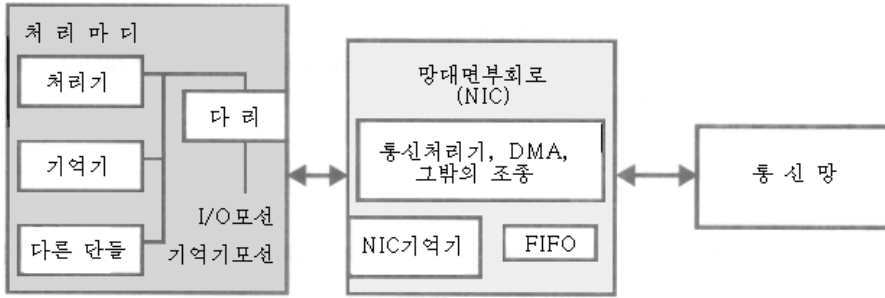



그림 7-16. 전형적인 통신하드웨어의 구조

대답

(1) 기가비트망은 하드웨어병목이다. 그래서 한계는 128MB/s이다.

(2) I/O모선은 하드웨어병목이다. 그래서 한계는 133MB/s이다. 이 병목을 없애기 위하여 강하게 결합(tightly-coupled)된 MPP들(실례로 Intel Paragon)은 NIC를 직접 기억기모선에 련결시킨다. 다음은 NIC가 병목으로 되며 대역너비는 200MB/s로 제한된다.

실례 7.4에서 준 한계들은 릉가할수 없는 하드웨어한계들이다. 실지로 달성할수 있는 대역너비는 흔히 더 낮다.

Stricker와 Gross[591]는 통신성능이 흔히 망대역너비보다 기억기대역너비에 의해 한정된다는것을 언급하였다. 실례로 많은 통신방법들은 제한된 기억기구역(DMA완충기라고 한다.)에 접근하기 위하여 DMA엔진을 요구한다. 사용자자료는 보내기전에 먼저 DMA완충기에 복사되여야 한다. 기억기복사대역너비는 보통 기억기모선의 최대대역너비보다 훨씬 더 작다.

표 7-4는 3가지 MPP들의 여러가지 구성요소들의 대역너비를 보여 준다.

표 7-4 3가지 MPP들에서 측정한 대역너비(MB/s)

MPP체계	기억기모선	복사	I/O모선	망	응용대역너비
Intel paragon	400	68	–	200	52
IBM SP2	2100	>120	80	40	35
Gray T3D	320	93	–	300	70

Paragon과 T3D에 대한 I/O모선항목들은 이 두 체계에서 NIC가 기억기모선에 직접 련결되므로 비여 져 있다. 응용대역너비는 응용사용자가 달성할수 있는것이다.

망대면부회로에 대한 서로 다른 구조들이 제안되였다. 그러나 여러개의 대중적인 특징들이 나타났다.

- NIC는 DMA기능을 가진다. 프로그람식I/O는 높은 처리기부가처리를 초래할뿐아니라 긴 지연과 낮은 대역너비를 가져 온다. 실제의 NIC들에서는 1~3개의 DMA엔진들이 사용되였다.
- NIC는 NIC처리기 또는 협조처리기, 극소형조종기, 적응기처리기라고 하는 자체의 처리기를 가진다. 마디안의 기본처리기를 **마디처리기** 또는 **호스트처리기**라고 부른다. NIC처리기는 DMA초기화, 파케트싸기/풀기, 보호검사 등에 필요하다.
- 많은 체계들에서 모든 DMA들은 마디처리기가 아니라 적응처리기에 의해 초기화된다. 이것은 자료가 마디처리기에 관계없이 움직일수 있다는 우월성을 가진다. 즉 마디처리기는 통신망으로부터 분리된다.
- NIC는 NIC코드를 기억하고 통보문을 림시로 완충시키는 기억기를 가져야 한다. 마디처리기는 프로그람식I/O를 통하여 이 기억기에 접근할수 있다. 현재의 체계들에서 이 NIC기억의 크기는 100KB에서 수메가바이트의 범위에 있다.

통신쏘프트웨어

쏘프트웨어부가처리는 흔히 현재의 클라스터들과 MPP들에서 통신시간을 좌우한다. 이로부터 효률적인 통신쏘프트웨어를 가지지 않으면 대단히 효률적인 망과 NIC를 가지고도 통신시간을 크게 줄일수 없다는 결론이 나온다.

쏘프트웨어부가처리는 주로 3가지 원인으로부터 생긴다.

- 쏘프트웨어는 여러개의 규약층들을 지나가야 한다. 이러한 형태의 부가처리를 줄이기 위한 일반기술은 규약구조를 간단하게 하는것이다.
- 통보문통신은 령복사규약(zero−copy protocol)을 호출하는 여러개의 기억기복사요구를 포함할수 있다.
- 통신쏘프트웨어는 통보문을 전송할 때 보호경계를 여러번 벗어 날수 있다. 이 문제를 처리하기 위하여 사용자공간에서 모든 통신을 완전히 수행하는 사용자수준기술들이 나타나고 있다.

일반적인 규약구조들을 그림 7−10에서 설명한다.

잘 알려 진 소케트대면부는 망파일체계, 원격절차호출, ftp, telnet, http 등과 같은 많은 분산체계응용들에서 사용되였다.

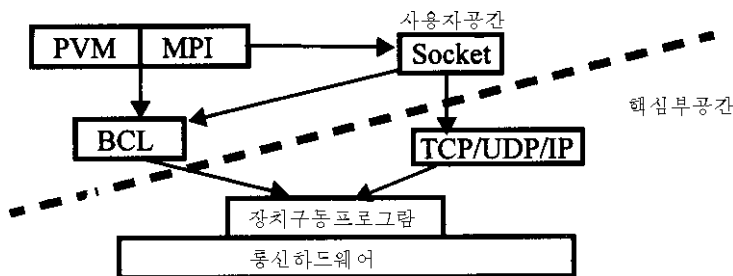



그림 7−17. 통신규약의 여러가지 구조들

PVM과 MPI와 같은 통신서고들은 소케트우에서 실현되였다. 송신때 통보문은 소케트층, TCP/IP(또는 UDP/IP)층, 구동프로그람 및 망하드웨어층의 순서로 통과한다. 수신단에서는 같은 처리가 반대순서로 반복된다.

소케트는 TCP/IP를 에돌아 저수준의 기초통신층(base communication layer, BCL)우에 직접 실현될수 있다. 한가지 실례는 BCL로서 Active Messages를 사용하는 Berkeley Fast Socket이다.

PVM/MPI도 socket/TCP/IP층들을 에돌아 BCL우에 실현할수 있다. BCL의 기본목적은 응용사용자들에게 가능한껏 생하드웨어성능만큼 제공하는것이다.

통신부분체계의 성능을 평가하는데서 MPI와 PVM, socket들의 성능평가는 BCL들의 성능평가보다 더 중요하다.

전체 통신시간의 큰 몫은 한 기억구역으로부터 다른 기억구역으로 자료를 복사하는데 소비된다. 기억기복사사건들의 수와 매 복사시간을 줄이는것이 중요하다. 리상적으로 령복사규약이 사용되여 통보문은 원천마디의 송신완충기로부터 목적지마디의 수신완충기에로 다른 기억기구역에서 또 다른 기억기구역에서 완충되지 않고 직접 이동하게 될것이다.

실례 7.5. IBM SP통신쏘프트웨어

IBM SP다중콤퓨터에서 통보문을 보내는데 포함된 기억기복사를 그림 7-18에서 설명한다. 전송될 통보문은 송신완충기에 기억되며 사용자응용에서는 보통 변수로 나타낸다.

실례로 사용자응용이 한마디의 배렬 A를 다른 마디의 배렬 B로 보내려 한다고 하자. 이때 A는 송신완충기를 나타내며 B는 수신완충기를 나타낸다. 내부적으로 관흐름완충기와 I/O대기렬들이 통신쏘프트웨어에 의하여 사용된다.

SP통신규약은 송신마디의 처리기가 자료를 송신완충기로부터 관흐름완충기로 복사한 다음 관흐름완충기로부터 출구대기렬로 복사할것을 요구한다. 수신마디의 처리기는 같은 동작들을 반대순서로 수행한다. 총체적으로 4개의 기억기복사연산들이 수행된다. 복사를 줄이기 위하여 SP는 그림 7-18에서 점선들로 보여 주는바와 같이 긴 통보들이 관흐름완충기를 에돌아 가도록 한다.

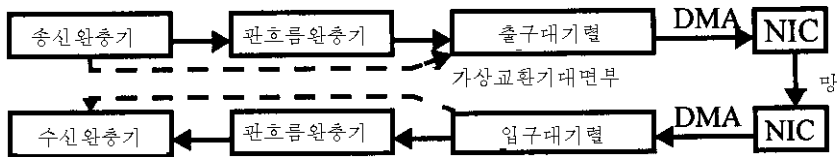



그림 7-18. IBM SP 통신쏘프트웨어에서의 자료이동

일반적인 통신쏘프트웨어에서 통보문넘기기는 송신측과 수신측에서 핵심부에 들어가고 핵심부를 떠나는데서 적어도 4번은 보호경계를 교차한다. 많은 현대의 가동환경들(실례로 RISC처리기와 유닉스)에서는 핵심부에 들어 가는데 적어도 10μs 걸린다. 그러므로 사용자공간에서 통보문전송을 완전히 수행하는것은 지연을 크게 줄일수 있다.

통신환경

통신부분체계의 설계에 영향을 주는 또 다른 요인은 부분체계가 작업하는 환경이다. 고찰하여야 할 문제점들을 표 7-5에 렬거한다.

표 7-5 통신환경의 문제점들

문제점	낮은 일반화	높은 일반화
응용	기술적계산	상업분산
과제	단일과제	다중과제
동질	SPMD	MPMD
스레드	단일스레드	다중스레드

상업적 및 망식/분산식응용들을 위하여 통신부분체계는 다중프로그람환경에서 TCP/IP와 같은 표준규약들을 효률적으로 지원하여야 한다.

전용MPP들우에서의 기술계산응용들은 통신부분체계가 많은 경우에 PVM과 MPI만을 지원하므로 더 단순하다. SPMD는 많은 기술계산응용들에서 사용되며 한편 상업 및 분산응용들은 흔히 MPMD이다.

이전의 MPP마디의 프로쎄스들은 단일스레드식이지만 다중스레드식프로쎄스들은 현대의 MPP들과 클라스터들에서 특히 SMP마디가 사용될 때 사용된다. 결정적인 문제점은 하나의 마디우에 하나 또는 여러개의 프로쎄스들이 있는가 하는것이다.

단일과제(또는 단일프로그람작성)는 이전의 MPP들에서 사용되였는데 여기서는 매 마디에 끝날 때까지 실행되는 하나의 프로쎄스만이 있다. Cray T3D/T3E에서 일감의 프로쎄스들은 계산도중에 나갈수 있으며 프로쎄스들이 다른 일감으로부터 바로 빈 마디들에로 들어 올수 있다. 일감의 프로쎄스들은 무리일정작성식(gang-scheduled)이여서 일감의 한 프로쎄스가 어떤 마디에 있을 때 일감의 다른 모든 프로쎄스들은 다른 마디들에 있을수 있다.

다중과제로 하여 서로 다른 일감들의 여러 프로쎄스들이 같은 시간에 하나의 마디우에 있을수 있다. 다중과제는 통신부분체계의 설계를 상당히 복잡하게 한다. 실례로 다음의 문제점들이 제기된다.

- NIC가 프로쎄스 P의 DMA완충기로부터 DMA를 통하여 통보문을 받고 있는 동안 그 완충기는 보충기억기공간을 요구하는 다른 프로쎄스 Q에 의해 바뀌여 진다. 그러면 NIC는 본래의 DMA완충기가 지금은 프로쎄스 Q의 자료를 가지고 있으므로 틀린 자료를 전송한다. 이 문제를 풀기 위하여 DMA완충기는 절대로 바뀌지 말아야 한다.
- 마디는 여러 프로쎄스들에 의해 시간공유되기때문에 임의의 프로쎄스는 임의의 시간에 비결정적으로 시간초과될수 있으며 프로쎄스가 얼마동안 일정이 작성되지 않고 남아 있겠는지 예측할수 있다. 일정이 작성되지 않은 프로쎄스에 보내진 통보문은 수신측을 차단할수 있다. 그리고 수신단의 혼잡에 의한 후과는 결국 망전체를 차단한다.

- 다중과제는 또한 보호를 보다 환기시킨다. 왜냐하면 여러 프로쎄스들이 NIC와 련관된 통신요구들, 대기렬들, 완충기들 등을 공유하기때문이다. 이 곤난은 사용자수준의 통신에 대한 지원과 혼합된다.

실례 7.6. IBM SP의 통신부분체계

IBM SP의 통신부분체계는 통신환경의 문제점들을 푸는데서 한가지 절충안을 제공한다. 그것은 계산과 상업/분산응용들을 다 지원한다. 병렬응용은 SPMD 또는 MPMD일수 있다. 매 마디는 워크스테이션과 꼭 같은데 다중과제가 가능하다.

고속HPS망과 표준IP, 독점적인 사용자공간(US)규약우에서의 통신에는 2개의 규약이 리용가능하다.

NIC부터 HPS까지는 한마디의 여러 프로쎄스들에 의해 공유될수 있다. 그러나 매 마디에서 오직 하나의 프로쎄스만이 US규약을 사용할수 있다.

통신봉사

봉사들이 제공되는것은 통신부분체계의 성능에 크게 영향을 줄수 있다.

실례로 BCL은 제한된 환경에서 대단히 한정된 봉사들만을 제공하지만 좋은 성능을 보여 준다. 그러나 이 BCL의 최상위에 구축된 MPI/PVM은 그 실현이 보충적인 봉사들을 제공하여야 하므로 불충분하게 수행된다.

다음의 봉사들이 통신부분체계에서 기대된다.

- **담보넘기기** 일단 하나의 Send함수호출이 귀환하면 통신부분체계는 일반상황에서 의도하는 목적지마디에 완전한 통보문을 넘기기한다. 통보문은 틀린 마디로 가지 않으며 통보문의 파케트는 하나도 잃어 지지 않는다. 믿음성이 약한 둘중의 하나선택은 최대의 노력이 드는 넘기기이며 여기서 부분체계는 최대로 노력하지만 담보는 제공하지 않는다.
- **흐름조종** 통신부분체계는 완충기넘침이나 교착, 엄중한 혼잡을 피하기 위하여 흐름조종을 수행하여야 한다. 특히 수신측 혼잡문제가 제기된다. 여러개의 마디들이 하나의 수신마디에로 동시에 통보문을 보내고 수신측 프로쎄스는 계산에 바쁘고 또 얼마동안 어떠한 수신정합도 실행하지 못한다고 가정하자. 들어 오는 통보문들이 수신측 완충기를 채우자마자 포화가 뒤로 퍼져 전체 망을 포화시킬것이다.
- **고장처리와 담보넘기기** 통신부분체계 특히 망에 고장이 일어 났을 때 그 고장을 프로쎄스하는 세가지 방법이 있다. 그것은 무시하거나 통보하거나 고치는것이다. 이써네트우에서 TCP를 사용하는 분산체계는 중간이 믿음직하지 않기때문에 마지막방법을 사용한다. 수신한 파케트에 대한 오유검사를 하고 오유가 발견된 후에 재전송된다.

  특수한 망을 사용하는 MPP들과 클라스터들에서는 망오유률이 흔히 매우 낮기때문에 첫 두가지 방법들이 사용될수 있다. 실례로 DEC기억기통로망은 다만 10^{-16}의 1bit오유률을 가진다. 즉 10^{16}bit 전송당 1개의 오유비트가 있다. 통신규약이 정확한 통보문이 정확한 목적지에 넘기기될것이라는것을 담보하면 담보넘기기가 제공된다.

- **순서식넘기기** 원천마디가 두개의 통보문을 같은 목적지에 보낼 때 목적지에서 수신연산의 실행은 첫번째 통보문을 수신한다

7．4．2．LogP통신모형

Culler et al.[178]은 병렬콤퓨터를 네개의 파라메터 즉 L, o, g, P로 특징화하는 LogP 모형을 제안하였다. 짧은 통보문은 한개 단어, 두개 단어 등이다.

Log P모형은 그림 7-19에서 설명하는바와 같이 통신알고리듬을 개발하는데 유용하다.

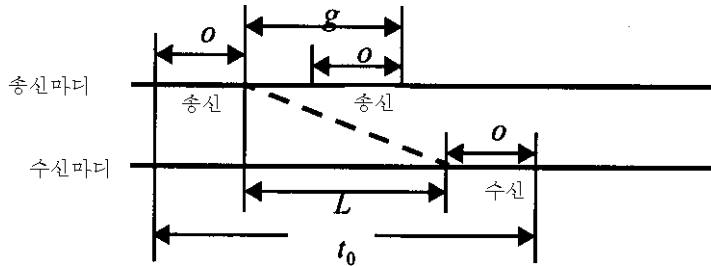



그림 7-19. Log P 모형에서 통보문의 통과

한 마디로부터 다른 마디로 한개의 단어를 통신하는 시간(즉 1.3.2의 체계기동시간 t_0)은 세개의 구간으로 다시 나누인다. 즉 $t_0 = o + L + o$ 이다. 첫번째 o는 송신부가프로쎄스라고 하는데 송신마디가 한개 단어를 망으로 주입하기 위하여 통보문보내기연산을 실행하는 시간이다. 두번째 o를 수신부가처리라고 하는데 수신마디가 망으로부터 1개 단어를 가져 오기 위하여 통보문수신연산을 실행하는 시간이다. 간단히 하기 위하여 두개의 부가처리는 같다고 가정하고 부가처리를 o라고 한다. 다시 말하여 부가처리 o는 마디가 한개의 단어를 보내거나 받는데 소비하는 시간의 길이이다. 이 시간동안 마디는 다른 연산들을 수행할수 없다(실례로 겹칩계산). 지연 L은 하드웨어적망지연 즉 한개의 단어를 송신마디로부터 수신마디로 망을 통하여 넘기기하는 시간이다. 간격(gap)파라메터 g는 한개 마디에서 2개의 련속적인 통보문송신 또는 수신사이에 요구되는 최소시간간격으로 정의된다. Log P모형은 후에 Log GP[22]로 확장되는데 여기서는 대역너비의 호상련관과 동등한 부피간격이라고 하는 파라메터 G가 첨가된다. 즉 $G=1/r_\infty$이다.

실례 7.7. 두가지 병렬콤퓨터의 LogP파라메터들

대부분의 현존 병렬콤퓨터들에서 부가처리 o는 지연 L에 비해 훨씬 더 크다. 그러나 바로 그 반대를 보여 주는 기계들도 있다. 이것을 표 7-6에서 보여 준다. 표에는 두가지 병렬콤퓨터의 Log GP파라메터들에 대한 근사값을 기입하였다.

표 7-6 두가지 병렬콤퓨터의 LopP파라메터들

기계	L	O	G	P	T_0	r_∞
IBM SP2	1.25μs	22.38μs	24μs	400	46μs	35MB/s
MeiKo CS-2	10μs	3.8μs	13.8μs	64	17.6μs	43MB/s

Log P모형은 통신의 병목들을 식별하는데 사용될수 있다. 최근에 Martin el al.[431]은 다음의 두가지 문제들을 처리하는 개선된 LogGP모형을 제기하였다.

네개의 파라메터(망지연 L과 처리기의 부가처리 o, 간격 g, 대역너비 1 /G)중

(1) 어느것들이 응용의 성능에 가장 크게 영향을 주는가?

(2) 그것들은 성능에 얼마만큼 영향을 주는가?

이것들은 중요한 문제들이다. 왜냐하면 통신부분체계의 설계자는 제한된 자원을 가장 중요한 파라메터를 개선하는데 사용하려고 하기때문이다.

Martin et al.은 32개 마디 워크스테이션클라스터에 대하여 10개의 병렬콤퓨터 성능평가기준프로그람들의 성능을 측정하여 그 영향을 연구한다. 이 프로그람들은 통보문넘기기병렬콤퓨터에 공유기억기환경을 제공하는 언어 Split−C로 실현된다. 시험가동환경은 지연과 부가처리, 간격을 독립적으로 조절할수 있도록 한다. 클라스터가동환경의 고장환경은 다음의 파라메터값들을 가진다.

L=5**us**, o=2.9us, g=5.8us, 1/G=38MB/s

연구는 여러개의 흥미 있는 점들을 밝혔다.

- 응용들은 부가처리에 가장 민감하다. 응용들은 부가처리가 2.9μs로부터 103μs로 증가할 때 50배까지 속도가 떠질수 있다. 통신응용들은 3~5배로 속도가 떠질수 있다. 응용의 실행시간은 첨가한 부가처리에 선형적으로 비례한다는것이 관찰된다. 물론 서로 다른 응용들은 서로 다른 결수를 가진다.
- 응용들은 간격에도 예민하다. 그러나 부가처리만큼 예민하지는 않다. 간격이 5.8μs에서 105μs로 변할 때 일부 응용들은 영향을 받지 않았으며 가장 나쁜 응용은 6배 속도가 떠진다. 응용의 실행시간은 간격에 선형으로 비례한다는것이 관찰된다.
- 대부분의 응용들은 망지연에 균등하게 예민하다. 최악의 경우 지연이 100us이상으로 증가될 때 속도저하는 4배이하이다. LogP모형에서 지연은 망하드웨어지연이지 체계기동시간 t_0은 아니다. 연구용만한 상업병렬콤퓨터에서 망지연은 10μs이하이다. 100μs의 지연은 너무 길다.
- 대부분의 응용들은 통신체계의 부피대역너비(bulk−bandwidth)에 균등하게 예민하다. 최악의 경우에 속도저하는 부피대역너비를 단지 1MB/s로 줄일 때 3배이하이다. 모든 성능평가기준들에 대한 측정된 성능은 대역너비를 15MB/s로 줄일때까지 같은 수준에 머무른다.

결과들은 부가처리가 가장 위험하며 다음은 간격이고 지연과 대역너비는 가장 위험이 적다는것을 보여 준다. 처리기부가처리의 중요성은 많은 사람들에게 인식되였으며 7.4.3에서 론의되는 모든 기술들은 부가처리를 줄이는데 리용된다. 그러나 7.4.3에서 론의되는바와 같이 SP3통신부분망설계는 서로 다른 지침을 따르고 첫번째 우선

권은 대역너비에 주고 있다. 이 연구에서 어느 정도 놀라운 결과는 15MB/s의 대역너비이면 충분하다는것이다. 이 연구는 병렬콤퓨터들을 위한 정당한 하나의 자료점을 제공한다.

다른 병렬콤퓨터성능평가기준들 또는 처리량처리, 상업응용들에 대해서도 같은 결론이 성립하는가는 연구해 보아야 한다. 연구는 처리기의 속도가 증가할 때 결과들이 어떻게 변하는가 하는것을 밝히지 않았다.

이 점은 11.5에서 제기된다.

7. 4. 3. 저수준통신지원

그림 7-17로부터 통신성능을 높이는데서 BCL이 부정적인 역할을 논다는것을 알수 있다. 이 부분에는 세개의 대표적인 BCL들 즉 SHRIMP VMMC (zero-copy)와 Fast Messages(1-copy), SP2 user-space(US)규약(2-copy)을 론의한다. 다른 BCL들에는 Active Messages, U-Net, MeiKo CS2, Cray T3D/T3E, DEC TruCluster/Memory Channel, Intel Paragon/TFLOP 등이 있다.

통신부분체계의 성능을 평가할 때 점대점통신성능을 세개의 수준에서 목록에 기입한다.

- 하드웨어수준성능은 통신하드웨어가 BCL에 제공할수 있는 최대성능이다. 이것은 하드웨어의 최대성능과 같지 않다. 후자는 전자의 자료를 리용할수 없을 때 사용된다.
- BCL최대성능은 응용에서 반드시 달성가능하지는 않다.
- 응용수준최대성능은 PVM이나 MPI, RCP를 사용하는 최대성능으로 나타난다.

SP2통신쏘프트웨어

IBM SP체계의 통신쏘프트웨어의 구조를 그림 7-20에서 보여 준다. SP2은 두개의 규약 즉 핵심부공간의 IP에 기초한 규약과 US라고 하는 사용자공간규약을 지원한다. 핵심부공간규약에 대하여 AIX망대면부구동프로그람(network interface driver, NID)이 IP를 통신적응기에 접속한다.

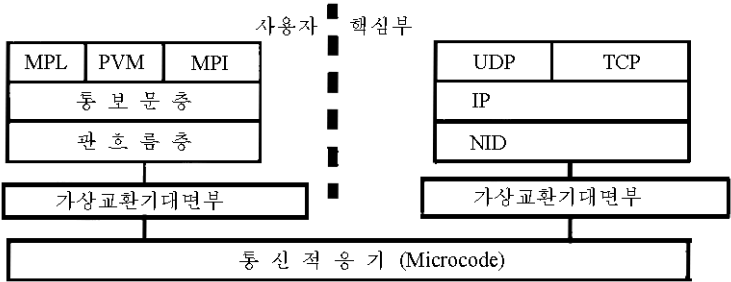



그림 7-20. SP통신쏘프트웨어

NID는 가상교환기대면부를 사용하는데 이것은 마디의 체계기억기의 두개의 대기렬과 적응기기억기의 몇개의 조종등록기들로 구성된다. 두개의 대기렬에서 하나는 입구를 위한것이고 다른것은 출구를 위한것이다. 대기렬은 적응기가 DMA자료를 넣고 꺼낼수 있는 체계의 DMA완충기를 제공한다.

두개의 갈라 진 가상교환기대면부가 있는데 하나는 IP규약용이고 다른것은 US규약을 위한것이다. IP에 기초한 규약은 표준적이며 한편 고성능을 위하여 설계된 US규약은 전용적이다.

표 7-10에서 볼수 있는바와 같이 US규약은 IP에 비해 성능을 크게 증가시킨다.

표 7-7　　SP2에서 점대점통신의 성능

	지연 t_0, μs	대역너비 r_∞, MB/s	최대절반길이, $m_{1/2}$B
하드웨어	<1	40	<40
MPL(US)	39	35.5	1385
MPL(UDP/IP)	277	10.8	2991

(MPL은 MPI와 비슷한 응용수준통보문넘기기서고이다.)

US규약이 하드웨어최대대역너비의 80%이상을 응용사용자에게 제공할수 있다는것이 중요하다. US규약은 다음의 사실들과 결론들에 기초하여 설계된다.

- HPS는 기술계산을 위한 통보문넘기기통신뿐만아니라 흔히 TCP/IP를 사용하는 상업응용들, 체계봉사들, I/O연산들(실례로 파일)을 지원하여야 한다. 그러므로 적응기는 US규약을 사용하는 단일처리와 IP를 사용하는 임의의 수의 프로쎄스들사이에서 공유되여야 한다.
- 3가지 통신성능파라메터들중에서 대역너비는 가장 중요하게, 지연은 두번째, 마디처리기의 부가처리는 제일 약하게 간주한다.
- 기본마디처리기(POWER2처리기)는 적응기(1860)안의 통신처리기보다 더 강력하다. 그러므로 통보문처리의 많은 몫을 마디처리기가 수행하도록 관심을 돌린다.
- 마디처리기가 프로그람식I/O를 통하여 적응기기억기에 접근하는것은 느리므로 그러한 접근을 제한하는것이 중요하다.

그림 7-20의 통보문층과 관흐름층은 그림 7-17의 기본통신서고(base communication library, BCL)를 형성한다.

통보문층은 몇개의 단순한 비차단식의 점대점통신서고함수들로 구성된다. MPL과 MPI, PVM의 더 높은 수준의 모든 통보문넘기기함수들은 이 요소들을 사용하여 실현된다.

관흐름층은 임의의 송신/수신프로쎄스쌍들사이의 믿음직하고 흐름조종된 순서화식바이트흐름을 제공하는 한쌍의 관흐름을 유지한다.

기본적인 송신-수신알고리듬은 그림 7-18, 그림 7-20, 그림 7-21을 참조하면 리해된다.

- 원천마디의 처리기는 자료를 송신완충기로부터 관흐름완충기로 복사하는 통보문층송신연산을 실행한다(만일 관흐름완충기가 리용불가능하면 송신을 차단한다.). 다음 통보문층은 자료를 관흐름완충기로부터 출구대기렬로 복사하기 위하여 관흐름층코드를 발송한다.
- 원천지의 적응기는 DMA에 의해 출구대기렬로부터 적응기로 자료를 옮긴 다음 그 자료를 망을 거쳐서 목적지의 적응기로 전송한다.
- 목적지의 적응기는 들어 오는 자료를 DMA에 의해 목적지마디의(가상교환기대면부안의) 입구대기렬로 전송한다.
- 목적지마디의 처리기는 자료를 입구대기렬로부터 관흐름완충기로 복사하는 관흐름층을 실행한다. 다음 관흐름층에서는 정합된 수신이 이미 기입되여 있으면 관흐름완충기로부터 자료를 최종적인 수신완충기로 복사하기 위하여 통보문층을 호출한다.

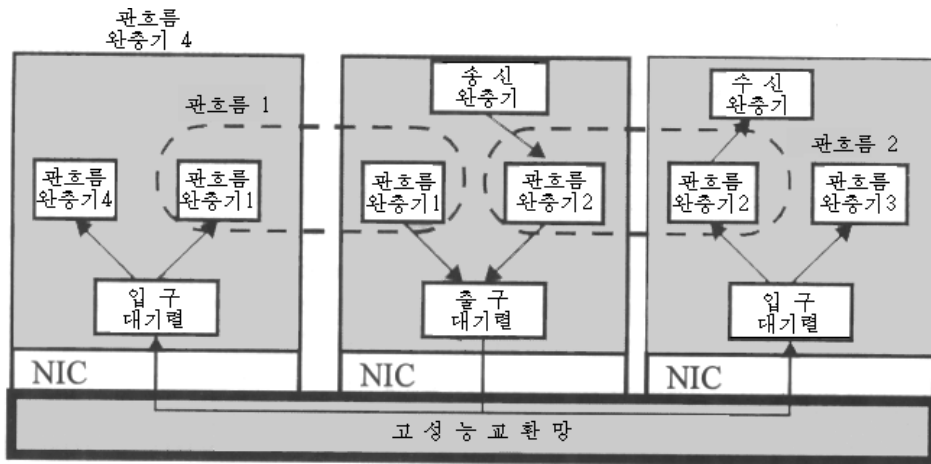



그림 7-21. SP2 통신부분체계에서 완충엔진리

마지막 결음은 정교한 완성을 요구한다.

목적지처리기는 망을 류출시키기 위하여 어떻게 관흐름을 실행하는가?

이것은 송신측의 혼잡이 통신성능을 크게 낮출수 있기때문에 대단히 중요한 물음이다.

IMB SP2에서는 오직 관흐름층만이 입구대기렬로부터 자료를 류출시킬수 있으며 마디처리기만이 관흐름층을 실행할수 있다. IBM SP2는 망을 류출시키는데 아래에서 론의하는 두가지 방법 즉 투표(polling)와 중단을 결합하여 리용한다.

투표방법에서 관흐름층은 통보문층에 의해 호출된다. 통보문층은 목적지의 프로쎄스가 수신연산을 실행함으로써 호출된다. 통보문층은 매 관흐름을 투표하기 위하여 관흐름층을 호출하며 관흐름층은 자료를 입구대기렬로부터 관흐름완충기로 이동시킨다.

정합통보문이 관흐름완충기에 도착하였으면 통보문층은 자료를 최종적인 수신완충기로 복사한다. 통보문이 도착하지 않았으면 수신연산은 후의 프로쎄스를 위하여 기입된다.

통보문이 목적지프로쎄스의 입구대기렬에 도착할 때 프로쎄스는 정합수신을 실행하지 않고 그밖의 다른것을 부지런히 하고 있을수 있다. 그때 통보문은 입구대기렬에 머물러 있는다.

관흐름층에서 전진을 담보하기 위하여 SP2 US규약은 망을 류출시키는 두개의 중단기구를 제공한다. 즉 관흐름층은 시계중단에 의해 주기적으로 호출되며 또 적응기완충기가 채워졌을 때 적응기중단에 의해 호출된다.

통신부분체계는 송신/수신프로쎄스들의 매개 쌍마다 하나의 관흐름을 구성하는데 이것은 그림 7-21의 2개의 Pipe Buf2와 같이 한쌍의 관흐름완충기로 구성된다. 매 프로쎄스마다 하나의 입구대기렬과 출구대기렬이 있다. 이 대기렬들은 교환가능하지 않는 기억기구역에서 유지되며 프로쎄스의 모든 관흐름들이 공유한다.

병렬프로그람이 마디당 1개씩 대응하는 *n*개의 프로쎄스들로 구성되면 송신마디에는 최대로 1개의 출구대기렬과 *n*개의 나가는 관흐름완충기가 있고 수신마디에는 최대로 1개의 입구대기렬과 *n*개의 들어 오는 완충기들이 있을것이다.

마디의 처리기는 계산코드와 통보문층, 관흐름층을 3개의 코드모두가 같은 사용자프로쎄스공간에 상주하는 협조적인 프로그람형식으로 실행한다. 때문에 체계호출이나 핵심부교차가 없다.

모든 DMA들은 적응기에 의해 초기화되고 수행된다. 모든 자료전송은 DMA에 의해 수행된다. 조종 및 상태정보만이 프로그람식I/O에 의해 마디의 처리기와 적응기사이를 통과한다.

SP2 US규약은 담보되고 순서화된 믿음직하고 흐름조종된 통보문의 넘기기를 제공한다.

적응기는 US통신연산들에서 사용한 국부프로쎄스의 ID들을 물리적인 주소로 변환하는 함수를 제공하며 통신이 배정된 구획내에서 진행되도록 담보한다.

통표규약(token protocol)은 흐름조종에 사용된다. 즉 자료는 목적지측에 리용가능한 공간이 있을 때에만 원천측관흐름으로부터 보내진다(즉 자료는 관흐름완충기로부터 출구대기렬로 복사된다.).

수신마디가 자료를 입구대기렬로부터 관흐름완충기로 류출시키기 위하여 관흐름층을 실행할 때 통표는 원천측에 해제된다. 시간초과와 재전송구조를 가지는 응답파케트들은 믿음직한 넘기기에 사용된다.

고속통보문

고속통보문(FM)은 클라스터들과 MPP들에서 낮은 지연과 높은 대역너비통신을 위하여 일리노이즈대학[480]에서 개발된 공공령역BCL이다. 이것은 Cray T3D와 Myrinet로 접속된 Sun Sparc워크스테이션들의 망우에서 실현되였다.

FM의 기본목적은 더 높은 수준의 패키지(실례로 소케트와 PVM, MPI)가 하드웨어의 한계에 가까운 통신성능을 실현할수 있도록 통보문층에서 충분한 기능성을 제공하는것이다.

T3D에서 FM은 130MB/s라는 하드웨어한계를 넘어 6.1μs의 지연과 112MB/s의 대역너비를 달성하였다.

표 7-10은 Myrinet로 접속한 SPARCstation의 20개 워크스테이션으로 된 클라스터에서 FM의 성능을 보여 준다. FM에 기초한 MPI실현(MPI-FM이라고 한다.)이 22MB/s라는 하드웨어최대값을 벗어 나 19μs의 지연과 17.3MB/s의 대역너비를 달성하였다는것이 중요하다. 사용자수준의 소케트대면부는 35μs의 지연과 11.3MB/s의 대역너비를 달성하였다.

표 7-8 고속통보문을 가지는 점대점통신의 성능

체계	지연 t_0 μs	대역너비 r_∞MB/s	반피크길이 $m_{1/2}$B
하드웨어	<1	23	12
FM	14	17.6	246
MPI-FM	19	17.3	296
Socket-FM	35	11.3	386

FM API는 Acive Message와 대단히 비슷한데 더 단순하고 몇개의 함수들만을 포함한다. 중요한 기본지령들을 표 7-9에서 보여 준다. FM은 통신에 FM을 사용하는 한개의 마디우에서 최대로 1개의 프로쎄스만이 실행된다는것을 전제로 한다.

표 7-9 고속통보문 응용프로그람작성대면부

FM요소들	의미
FM_initialize()	고속통보문을 초기화한다.
FM_send(dest,h,buf,size)	기억기로부터 하나의 긴 통보문을 보낸다.
FM_send_4(dest,h,i0,i1,i2,i3)	등록기들로부터 하나의 4개단어통보문을 보낸다.
FM_extact()	수신된 통보문을 추출하고 조종기를 호출한다.

함수 FM_initialize는 FM이 통신에 사용되기전에 즉 임의의 FM상수들이 대역변수들에 배정되기전에 그리고 임의의 통신함수가 호출되기전에 모든 마디의 프로쎄스가 호출하여야 하는 첫번째 함수이다. 이 함수는 FM서고가 사용하는 대역변수들을 초기화하며 통보문대기렬과 DMA완충기를 설치하고 마디 ID를 배정한다. 이것은 또한 장벽동기화를 수행한다.

FM_send_4(dest ,h, i0, i1, i2, i3)는 조종기를 호출하고 그것을 4개의 32bit정수인수 i0, i1, i2, i3과 송신지의 마디 ID로 통과시킴으로써 한개의 4개단어통보문을 목적지마디 dest로 보낸다.

FM_send(dest, h, buf, size)함수는 FM_send_4와 비슷하지만 기억기위치 buf에서 sizebyte의 통보문을 목적지마디 dest로 보낸다. 조종기 h는 buf와 size, 송신마디의 ID를 인수로 하여 호출된다. M_extract()호출은 모든 미정의 통보문들을 처리하며 그것들의 조종기들을 호출한다.

FM_send나 FM_send_4는 조종기가 호출될것이라는것만을 명기한다는데 주의하여야 한다. 실제적인 조종기의 호출은 FM_extract가 한다. 만일 통보문이 리용가능하지 않으면 FM_extract는 빈 함수호출과 같이 즉시에 귀환된다.

조종기는 임의의 계산을 포함할수 있다. 특히 조종기는 FM함수들을 호출한다. 조종기가 끝날 때까지 남은 자원이 없으면 교착이 생길수 있다. FM의 사용자는 교착을 막아야 할 책임을 진다. 하나의 간단한 방법은 강제로 조종기가 국부계산만을 수행하도록 하는것이다.

실례 7.8. FM기본지령들을 사용하는 통보문넘기기프로그람

다음의 SPMD코드는 마디 0의 위치 Buf에서 Sizebyte의 통보문을 어떻게 마디 1의 위치 A로 보내는가를 보여 준다.

```
#include "fm.h"
// 통보문을 위치 A로 옮기는 조종기 H를 정의한다
main()
{…
   FM_initialize();…
   If (FM_NODEID==0)
          FM_send(1, H, Buf, Size);
   else
           FM_extract();…
}
```

여기서 FM_NODEID는 FM_initialize로 배정한 마디 ID를 나타낸다. 서로 다른 마디들은 서로 다른 ID를 가진다.

FM통신규약은 그림 7-22에서 설명한다. 4개의 FIFO대기렬들이 사용된다. Myrinet적응기기억기는 하나의 송신대기렬과 하나의 수신대기렬을 포함한다. 마디의 핵심부기억기는 절환불가능한 DMA구역에 큰 수신대기렬을 포함한다. 거부대기렬(reject queue)은 흐름조종을 위하여 사용자공간의 기억기에 상주한다.

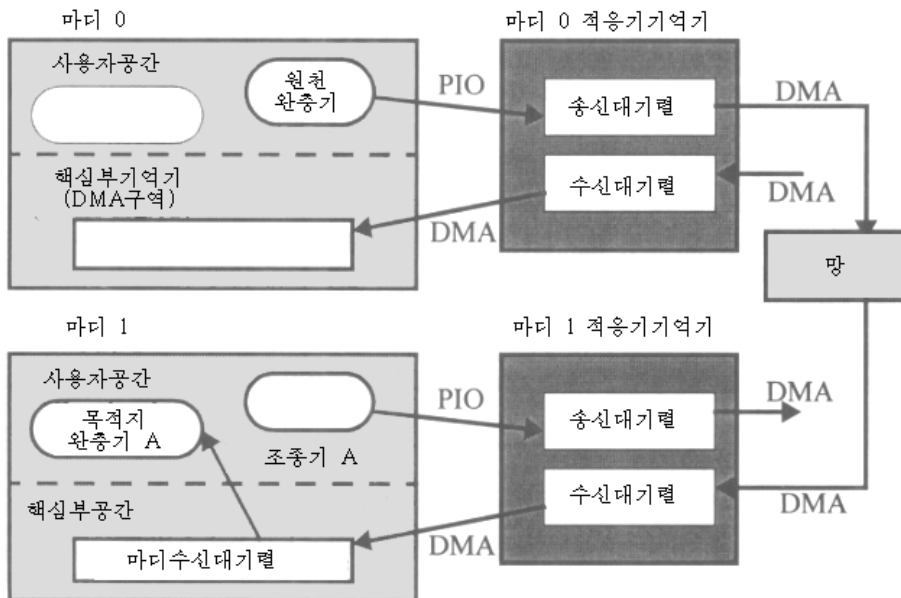



그림 7-22. Fast Message용통신규약(핵심부공간의 DMA 구역)

FM은 80Mb/s Myrinet로 접속한 SPARCstation의 20개 워크스테이션들의 클라스터우에서 실현되였다. Myrinet NIC적응기는 128KB SRAM을 포함하며 워크스테이션마디의 I/O모선(SBus)에 약하게 결합된다. 그것은 3개의 DMA기판을 포함하는데 하나는 적응기로부터 호스트에로 전송하고 둘은 적응기와 망사이에서 전송한다.

FM은 사용자수준통신층이다. 매 마디가 FM_initialize를 실행한후 적응기기억기와 마디의 수신대기렬은 사용자수준에서 공개된다. 다음의 FM_send(FM_send_4)와 FM_extract는 핵심부에 들어 갈 때 보호경계와 교차하지 말아야 한다.

실례 7.8로 되돌아 가자.

마디 0이 하나의 통보문을 마디 1로 보내려고 FM_send(1, H, Buf, Size)를 호출하면 다음의 걸음순서로 실행된다.

먼저 마디 0의 처리기는 Buf로부터 자료를 꺼내서 파케트들을 묶은 다음 프로그람식 I/O에 의하여 그것들을 적응기의 송신대기렬에 직접 기억한다. 남은 자료의 이동은 모두 DMA를 거친다. 파케트들은 마디 0의 적응기에 의해 망으로 주입된다.

파케트가 마디 1에 도착하면 그것은 적응기의 수신대기렬에 넣어 지며 후에 마디의 수신대기렬에 넣어 진다.

이 DMA들중 아무도 마디의 처리기를 포함하지 않는다. 특히 적응기로부터 마디의 수신대기렬에로의 마지막 DMA는 목적지의 프로쎄스가 일정이 작성되지 않는 동안 일어날수 있다. 이 분리는 수신측의 혼잡을 막게 해준다.

FM_send와 FM_send_4는 차단식송신이다. 하나의 전송함수호출이 귀환할 때 FM은 Buf가 재사용될수 있음을 담보한다.

수신마디(마디1)는 마디의 수신대기렬이 다 차지 않도록 FM_extract를 충분히 자주 실행하여야 한다. 매개 FM_extract프로쎄스들은 마디의 수신대기렬로부터의 모든 미정의 통보문들을 처리하며 관련된 프로쎄스자료를 포함한다.

실례 7.8에서 마디 1에 의해 실행된 FM_extract는 마디 0으로부터 보내온 통보문을 처리하며 통보문자료를 사용자기억기공간으로 옮기는 조종기 H를 포함한다.

Myrinet에 대한 FM

Myrinet에 대한 FM은 Myrinet가 10^{-15} 이하의 1bit오유률로서 고도로 믿음직하기때문에 순서화되고 흐름조종을 가지며 담보된 넘기기를 제공한다. 그러나 고장처리는 무시한다.

통보문의 순서는 모든 대기렬들과 Myrinet망완충기들이 FIFO라는 사실과 임의의 원천마디로부터 임의의 목적마디에로 유일한 경로가 있다는 사실, 흐름조종을 위한 규약부분과 담보된 넘기기가 통보문의 순서바뀜을 일으키지 않는다는 사실로부터 담보된다.

Myrinet망은 wormhole경로정하기를 사용한다.

출구포구가 바쁠(작업중일 때) 때 파케트들은 망완충기에서 차단된다. 50ms이상 차단된 파케트는 버려 진다. 왜냐하면 FM은 마디와 망을 분리하며 수신마디의 적응기는 수신측의 프로쎄스가 계산중에 있거나 바뀔 때조차도 목적지마디의 수신대기렬이 다 차지 않는 한 망을 류출시킬수 있기때문이다. 그러므로 목적지의 수신대기렬이 다 차지 않는 한 파케트는 버려 지지 않으며 담보된 넘기기가 달성될것이다. FM에서 이것은 점대점흐름조종방법에 의해 달성된다.

매 송신처리에는 신용이라고 하는 마디의 수신대기렬의 한 부분이 배정되는데 모든

부분들의 합은 전체 수신대기렬의 크기를 초과할수 있다. 송신처리는 송신측이 여전히 신용을 가지고 있을 때만 수신마디에로 통보문을 보낼수 있다.

수신처리가 그 마디의 수신대기렬로부터 하나의 파케트를 지울 때 그것은 신용을 송신처리로 돌려 보낸다. 어떤 리유로 하여 수신측의 프로쎄스가 수신대기렬로부터 파케트들의 지우기를 무시하면 결국에는 모든 전송들이 차단된다.

그러나 이미 전송된 파케트는 목적지마디의 수신대기렬에 리용가능한 공간이 있으면 버려 지지 않는다는것이 담보되였다.

SHRIMP VMMC

Princeton 의 SHRIMP 과제는 가상기억기식통신(virtual memory-mapped communication,VMMC)을 제기하였는데 이것은 령복사통신규약을 제공하는 얼마 안되는 BCL들중의 하나이다. 비슷한 방법을 리용하는 다른 BCL들에는 기억기통로망을 가지는 Meiko CS-2와 DEC TruCluster가 있다. 이 모든 체계들은 새로운 기억기관리요구들을 수용하기 위한 OS원천코드의 수정을 요구한다.

VMMC기구는 Myrinet로 호상접속된 PC클라스터와 전용통신하드웨어로 호상접속된 PC클라스터우에서 실현되였다. BCL수준의 성능은 표 7-10에서 보여 주는바와 같이 매우 강력하다.

표 7-10 VMMC를 가지는 점대점통신의 성능

방법	지연 $t_0,\mu s$	대역너비 r_∞,MB/s	최대절반길이 $m_{1/2}$,B
하드웨어	5	110	550
VMMC	9.8	108	1058
VRPC	33	33	1089

SHRIMP체계에서 마디는 PCI모선을 통하여 Myrinet에 접속된다. Myrinet와 PCI모선은 최대대역너비가 각각 160MB/s, 133MB/s이다. 그런데 110MB/s는 통신하드웨어가 VMMC로 달성할수 있는 최대대역너비이다. VMMC는 하드웨어대역너비의 98%이상을 달성한다. VRPC행은 VMMC에 기초한 RPC의 성능을 보여 준다.

성능자료들은 VMMC우의 MPI/PVM에는 전혀 리용가능하지 않다. 그러나 VMMC와 같은 통신방법은 좋은 MPI/PVM성능을 가져 올수 있다.

이것을 론증하기 위하여 표 7-10에서 VMMC와 비슷한 방법을 사용하는 DEC TruCluster의 성능을 보여 준다. TruCluster체계의 BCL을 UMP(Universal Message Passing)라고 부른다.

VMMC의 훌륭한 성능은 주로 두가지 기술 즉 보호된 사용자수준통신과 령복사규약에 근원을 두고 있다. VMMC기구는 그림 7-23에서 설명한다.

담보된 3개의 쏘프트웨어토막들 즉 VMMC데몬과 VMMC장치구동프로그람, VMMC를 실현하고 Myrinet망위상을 받아 들인 적응기조종프로그람이 매개 마디우에서 사용된다.

실례 7.8로 되돌아 가자.

마디 0이 통보문을 마디 1에 보내기 위하여 FM_send(1, H, Buf, Size)를 실행하면 다음의 걸음순서로 실행된다.

표 7-11 DEC TruCluster기억기통로망을 가지는 점대점통신의 성능

방법	지연 t_0, μs	대역너비 r_∞, MB/s	반첨수길이 $m_{1/2}$, B
하드웨어	2.9	64	186
UMP	5.8	61	354
MPI	6.9	61	421
PVM	8	43	344

먼저 마디 0의 처리기는 Buf로부터 자료를 꺼내서 그 자료를 파케트로 싼 다음 프로그람식I/O에 의해 그것들을 직접 적응기의 송신대기렬에 기억한다. VMMC데몬들은 송신마디와 수신마디사이의 중요한 관계를 확립하기 위하여 이써네트우에서 호상대화한다. 다음 마디 0의 프로쎄스는 자기 원천지완충기의 통보문을 마디 1의 다른 프로쎄스의 목적지완충기로 직접 전송할수 있다.

Fast Message와 IBM SP2에서와 같이 하나 또는 그이상의 핵심부공간을 통과하여 DMA완충기로 갈 필요가 없다. Fast Message와 IBM SP2에서 적응기는 통보문완충기의 물리주소를 보고 VMMC에서는 가상주소를 본다. 그러므로 적응기는 주소변환을 수행하기 위한 페지표들을 유지하여야 한다. 페지표들과 보호기구는 수입수출단계에 설치된다.

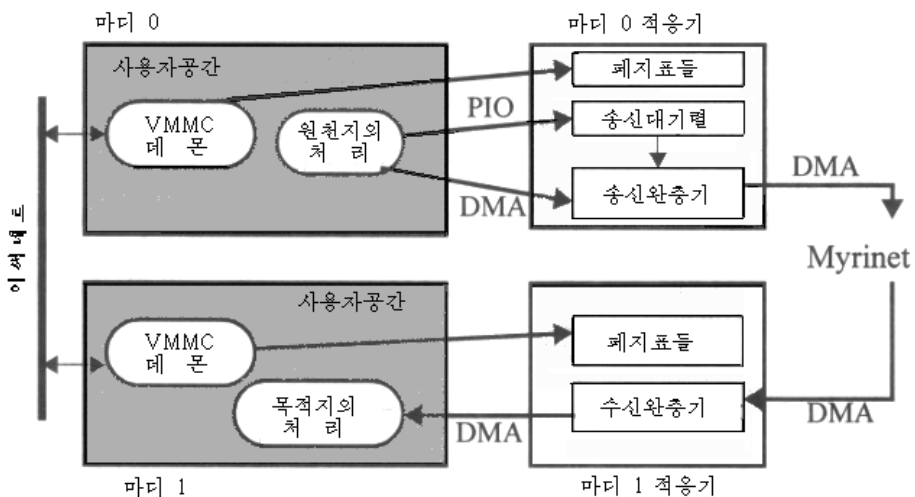



그림 7-23. SHRIMP VMMC의 통신규약

7. 4. 4. 통신알고리듬

7.3.3에서는 점대점통신을 효률적으로 지원하는 기술들에 대하여 론의하였다. 이 부분에서는 정규적인것보다 비정규적인것으로 한층 더 분류되는 집중통신들을 어떻게 지원하는가를 론의한다.

세대확장가능성과 종적확장가능성의 리유로 하여 병렬콤퓨터사용은 통신알고리듬을 개발하는 공통의 기계모형을 사용하는데로 지향되고 있다. 이 모형은 *n*개의 마디들로 구성되며 임의의 마디는 *m*byte의 점대점통보문을 임의의 다른 마디로 $t_0 + m \times t_c \mu s$ 동안에 직접 전송할수 있다.

여기서 t_0과 t_c는 각각 체계기동시간과 byte당 시간이다. 다시 말하여 호상접속위상은 무시된다. 적게 사용되는 다른 모형은 마디가 통보문을 모든 마디들에 방송할수 있다고 가정하는데 이것은 이써네트접속된 체계들의 추상화이다.

정규통신

실례 2.7에서 론의한바와 같이 많은 형태의 정규통신들이 있다. 그러한 통신들을 2개의 부류로 더 분류할수 있다.

뿌리통신(rooted communication)은 전체 대 1 또는 1 대 전체 통신이며 여기서 하나의 마디가 뿌리마디로 된다. 실례로 방송과 집합(gather), 산란(scatter), 축소(reduction)를 들수 있다.

동질통신은 명확한 뿌리마디의 개념이 없는 한가지 집체통신이다. 대신에 연산은 모든 마디들에서 같게 서술되여야 한다. 실례로서 전체 교환(total exchange), 장벽, 결합(여기서 모든 마디는 축소합을 받아 들인다.)이 있다.

아래에서는 전체 교환이 어떻게 실현되는가를 론의한다. 아래에서 잠간 소개하는바와 같이 전체 교환을 위한 3개의 주요알고리듬이 있다.

(1) 직접교환알고리듬은 Take[604]가 처음으로 제기하였는데 매우 간단하고 큰 통보문들에 대해서는 리론적으로 선택적이다.
(2) 표준교환알고리듬은 Johnson과 Ho[354]가 제기하였는데 짧은 통보문에서 잘 동작한다.
(3) 다단완전교환알고리듬은 Bokhari[99,100]가 제기하였는데 직접교환과 표준교환알고리듬 량쪽의 우월성을 결합한다.

이 알고리듬들은 모두 본래 하이퍼립방체다중콤퓨터들을 위하여 제기되였다. 그러나 Bokhari[100]는 다단알고리듬이 IBM SP2과 Intel Paragon, Meiko CS-2와 같은 비하이퍼립방체기계들에서도 잘 수행된다는것을 보여 주었다.

아래에서 직접 및 표준알고리듬들을 설명한다.

실례 7.9. 완전교환실현을 위한 직접교환방법

어떤 다중콤퓨터가 하이퍼립방체망으로 호상접속된 $n=2^d$개의 마디들로 이루어 진다고 하자. 매 마디는 함수 myid()를 호출하여 얻을수 있는 하나의 ID를 가진다. 초기에 마디 $i(0 \le i < n)$는 $i:0, i:1, i:2, \cdots, i:n-1$로 표시한 n개의 자료블로크를 포함한다.

완전교환의 마지막에 자료블로크 $i:j$는 마디 j안에 있어야 한다. 직접교환알고리듬은 다음과 같다.

여기서 매 마디는 같은 코드를 실행한다.

```
i=myid();
for(k=1;k<n;k++){
    destination= i ｜k;                    /*  i 배타합  k  */
    send block(i:destination) to destination node
}
```

이 알고리듬은 3가지 우월성을 가진다. 즉 그것은 대단히 간단하고 최소의 자료량을 전송하며 배타합연산은 동기차원순서경로정하기(synchronous dimension−order routing)가 사용되면 초평면에는 련결경쟁이 없다는것을 담보한다. 직접교환알고리듬이 최소의 자료량을 전송한다 해도 그것은 모두 $(n-1)(n-1)$번의 송신/수신을 실행하여야 하며 매 통보문은 1개의 블로크를 포함한다. 짧은 통보문에서 체계기동부가처리의 루적이 높을수 있다.

실례 7.10. 전체 교환을 위한 표준교환알고리듬

표준교환알고리듬은 여러개의 짧은 통보문을 큰 덩어리로 결합하며 매 마디는 log n개 통보문만 전송하면 된다. 이것은 체계기동부가처리의 루적을 줄인다.

매개 통보문은 $n/2$개의 블로크를 포함한다.

그리하여 전송된 전체 자료량은 $(n^2 \log n)/2$블로크이다. 그 알고리듬은 다음과 같다.

```
I=myid();
for(k=d;k>=0;k−−){
if(bit k of i is 1) superblock=block(i:0)to(i:n/2−1);
else superblocks=blocks(i:n/2)to(i:n−1);
destination=i｜2^k;                    /*  i 배타합 2^k */
send superblock to destination node
shuffle blocks locally          }
```

비정규통신

비정규통신들의 추상화는 정의 1.6에서 서술한 h원관계이다. 매 마디는 최대로 h개

의 단어들을 송신하며 최대로 h개의 단어들을 수신한다는것을 안다면 h관계이고 $h-1$ 관계는 아니다.

단순화한 h관계문제와 Badev et al.[51]의 해답을 론의한다.

그것들의 해답은 일반h관계문제에 적용할수 있다. N개 원소로 된 모임 S가 초기에 n개의 마디들속에 고르롭게 분포되여 어느 마디도 N/n개 이상의 마디를 가지지 않는다고 가정하자. 매 원소는 쌍 ＜data,dest＞이며 여기서 data는 마디 dest로 로정을 잡는다. 임의의 마디는 최대 h개 원소들의 목적지로 된다. 즉 N은 n으로, h는 n으로 나눈다.

Bader et al.의 알고리듬은 본래 Leslie Valiant[623]이 제기한 방법에 기초하고 있다. 착상은 비정규통신이 망경쟁을 줄이기 위하여 2단계절차로 실현되여야 한다는것이다.

첫 단계에서 매 마디는 자기의 통보문을 하나의 임의의 마디로 보낸다. 두번째 단계에서 통보문은 최종적인 목적지로 전송된다. 알고리듬은 다음의 걸음들을 가진다.

(1) 매개 마디 P_i는 n개의 저장함(bin)을 유지한다. 마디의 요소들은 n개 저장함중의 하나에 다음과 같이 배정된다. 원소들은 하나씩 주사된다. 원소는 그것이 목적지가 j인 첫번째 원소라면 저장함 $(i+j)\bmod n$에 배치된다. 그렇지 않으면 $(b+1)\bmod 2$에 배치한다. 여기서 b는 목적지가 j인 마지막원소가 배치된 저장함이다.
(2) 마디들은 전체 교환통신을 수행한다. 여기서 마디 Pi는 저장함 j의 내용들을 마디 P_j로 보낸다. 매개 저장함은 $N/n^2+n/2$개이하의 원소들을 포함한다.
(3) 매 마디는 자기의 최종적인 목적지들에 따라서 원소들을 n개의 저장소들에 재배렬한다.
(4) 마디들은 전체 교환통신을 수행하여 마디 P_i가 자기의 저장함 j의 내용을 마디 P_j로 보낸다. 매 저장함은 $h/h+n/2$개이하의 원소들을 포함한다.

Bade et al.은 이 알고리듬이 $O(h+t_0+(h+n^2+N/n)t_c)$라는 병렬시간복잡성을 가진다는것을 보여 주었다. 그들은 또한 여러 MPP들우에서 자기들의 알고리듬이 다른 현존 알고리듬들보다 더 빠르다는것을 보여 주는 실험자료들도 제기하였다. 특히 그들의 두단계알고리듬은 하나의 마디가 하나의 원소를 직접 그것의 최종목적지로 보내는 단일단계알고리듬들보다 더 빠르다.

그들의 h관계알고리듬은 통보문넘기기기계들을 위한 효률적인 병렬정렬알고리듬을 개발하는데 사용된다.

7. 5. 참고문헌주해와 련습문제

Solaris스레드의 개념과 실현은 Sun의 출판물[132,583,601]들에 서술되였다. 류사한 스레드기술들이 상업조작체계들[193,340]에서 사용되였다.

다중스레드는 자바프로그람작성언어의 하나의 특징이다. Lazy Threads approach[270]는 스레드창조를 순차적인 절차호출만큼 효률적으로 진행하려고 시도하고 있다. Click 스레드서고[94]에서 work-stealing일정작성은 스레드를 제한된 공간과 시간, 통신요구들로 작성된다. Nexus실시간체계는 통신과 스레드화를 통합하는 효과적인 방법을 보여준다.

Lamport의 1983년 론문[390]은 호상배제와 Lamport의 애완용동물문제를 제기한다. fetch-and-add와 결합의 착상은 Gottliele et al.[276]이 제기하였으며 한편 Kruskal et a.l[380]은 형식적인 취급을 준다. 쏘프트웨어결합기술은 Goodman et a.l[272]에 의해 제기되였다.

일반적인 잠금동기화의 부족점은 Herlihy et al.[306,307]과 Stone et al.[589]에 의해 론의되였는데 여기서 기다림 없는 동기화와 트랜잭션기억기, Oklahoma갱신이 제안되였다.

또 다른 하드웨어동기화구성은 Goodman et al.[272]에 의해 제기된 Queue-On-Lock-Bit 또는 QOLB이다. QOLB는 IEEE SCI표준에서 실현되였다. Kagi et al.[358]은 최근에 QOLB가 다른 잠금방법들을 릉가한다는것을 보여 주며 현재의 CC-NUMA기계들에서 쏘프트웨어로 실현될수 있다는것을 보여 준다.

Xu와 Hwang[654,658]은 프로그람작성자의 관점으로부터 잠금의 부족점을 론의하고 그 문제를 처리하는 일관성구역이라고 불리우는 새로운 언어구문을 제기하였다. 일관성구역은 12.3.5에서 더 론의된다.

동기화에서의 최근의 발전에 대하여서는 Mellor-Grummey와 Scott[44], Anderson과 Moir[32], Kontothanassis et al.[374]를 보시오.

TCP/IP조는 Comer의 책 [163]에서 상세히 서술된다.

현존 MPP들의 통신부가처리는 Karamcheti와 Chien[360]에서 해석된다. IBM SP2통신부분체계는 Snir et al.[574]에서 서술된다.

Illinois Fast Message(FM)은 Lauria와 Chien[394], Pakin et al.[480]에서 서술된다. Princeton SHRIMP에서의 통신속도증가노력은 Blumrich et al. [95, 96, 215, 216, 237, 342]에서 서술된다.

고속통신에 대한 다른 취급들은 Active Message[179, 426, 535, 626, 628], Myrinet[97], Unet[627]을 포함한다.

LogP모형은 Culler et al.[178]에 의해 제기되였는데 후에 LogGP모형 [22]로 확장되였다.

정규통신을 위한 효률적인 알고리듬들은 Bala et al.[59], Bokhari[100], Bruck et al.[112], Johnsson과 HO[354, 355]에서 찾을수 있다. 비정규통신기술은 Leighton[402]와 Bader et al.[51]에서 론의된다.

문　제

문제 7.1. Solaris스레드와 관련된 다음의 물음에 대답하시오.

(1) 두 수준구조(즉 스레드들과 스레드를 대신하는 LWP들)를 사용하는것이 가지는 두가지 우점은 무엇인가? 간단히 설명하시오.

(2) ZOBMLE스레드사용의 우점은 무엇인가? 수집스레드는 언제 실행되는가?

문제 7.2. 스레드와 프로쎄스의 수준에서 병렬처리를 비교하시오.

(1) 구체적인 실례들을 사용하여 다중스레드가 처리수준에서 병행개발에 비하여 리로운 4가지 리유를 설명하시오.

(2) 하나의 역실례를 주어 병렬성능에 관한 다중스레드가 적합치 않은 상황을 설명하시오.

문제 7.3. 잠금과 자물쇠해제를 사용하여 다음의 병렬코드를 고찰하시오.

```
parfor(i=0;i<n;i++)
{
   noncritical section
   lock(s);
          critical section
   unlock(s);
}
```

noncritical section은 T_{ncs}초, critical section은 T_{cs}초, lock(s)는 t_{lock}초 걸린다고 하자. Unlock(s)부가처리는 무시해도 좋다. 대응하는 순차코드를 수행하는데 $n(T_{ncs}+T_{cs})$초 걸린다. 해답에서 병행부가처리는 무시해도 된다.

(1) 전체 병렬실행시간은 얼마인가?

(2) N개의 처리기를 사용할 때 속도증가는 얼마인가?

문제 7.4. 두개의 공유변수 Ticket와 Turn에 접근하는데서 경쟁을 줄이기 위하여 7.2.4부분의 입장권알고리듬은 여러가지 기술들을 제기하였다.

(1) 그러한 기술 3가지를 간단히 서술하시오.

(2) n=4개의 처리기들이 있고 공유기억기의 매개 읽기 또는 쓰기는 100주기 걸리며

critical section을 한번 실행하는데 1000주기 걸린다고 가정하자. Delay함수는 처리기가 $k \times$(Myrinet−Turn)주기동안 기다리도록 한다.

다른 모든 연산들은 0시간 걸린다. K=100일 때 다음의 프로그람을 실행하는데 얼마만한 주기가 요구되는가?

```
int  Ticket=0,  Turn=0                    //  공유변수를 0으로 초기화
parfor(i=0;i<n;i++)                       //  n개의 프로세스들이 있다.
{int  Myrinet;                            //  국부변수
   Myrinet=FAA(Ticket,1));{               //  이 세개의 행은
   while(Myrinet!=Turn)                   //  잠금(lock)연산과
      Delay(Myrinet_Turn)        ;        //  동등하다.
   critical  section
   Turn=Turn+1;                           //  잠금해제(unlock)
}
```

(2)에서 실행시간이 최소로 되는 k의 선택값은 얼마인가? 근거를 가지고 그 값을 정당화하시오.

문제 7.5. 7.2.4에서 서술한 고속잠금기술의 사용에 대하여 다음의 문제들을 푸시오.

(1) 지수감소를 가지는 test_test_and_set잠금알고리듬을 작성하시오.

(2) 문제 7.4 (2)와 류사한데 배렬련결식알고리듬을 사용하는 호상배제프로그람을 작성하시오

문제 7.6. TCP/IP조에 대한 다음의 개념들을 설명하시오.

(1) 접속지향규약 대 비접속규약

(2) TCP 대 UDP(IP TCP와 UDP의 기능적차이에 초점을 두시오.)

(3) 소케트대면부

문제 7.7. 능률적인 통신에 대한 다음의 개념들을 설명하시오.

(1) 마디의 처리기 대 NIC처리기. 그것들은 무엇에 사용되는가?

(2) 보호방식사용자수준통신. 그 우점은 무엇인가?

(3) 1복사규약 대 령복사규약. 매 규약에서 송신시작으로부터 대응하는 수신의 끝까지 얼마만한 기억기복사연산들이 수행되는가 설명하시오.

(4) 담보넘기기 대 순서식넘기기

문제 7.8. 규약처리와 기억기복사, 보호경계교차는 통신쏘프트웨어부가처리의 3가지 원천이다. 이 세가지 형태의 부가처리를 줄이기 위하여 SHRIMP와 Fast Message, SP2 US 규약들이 어떻게 노력하였는가를 대조하는 간결한 표를 만드시오. 보다 상세한 설명본문으로 그것들의 우점과 부족점을 비교하시오.

문제 7.9. 8개의 마디로 된 2진하이퍼립방체다중콤퓨터가 점대점체계기동시간 t_0=50us와 점근(asymptotic)대역너비 r_∞=100MB/s를 가진다. 2진하이퍼립방체호상접속은 차원순서경로정하기를 사용한다. 다른 모든 시간은 무시한다.

(1) 직접교환알고리듬으로 100MB배렬을 전체 교환하는데 걸리는 시간은 얼마인가?

(2) 표준교환알고리듬으로 100MB배렬을 전체 교환하는데 걸리는 시간은 얼마인가?

(3) 직접교환알고리듬으로 1MB배렬의 전체 교환 100개를 수행하는데 걸리는 시간은 얼마인가?

(4) 표준교환알고리듬으로 1MB배렬의 전체 교환 100개를 수행하는데 걸리는 시간은 얼마인가?

문제 7.10. 4개 마디체계에서 다음의 *h*관계를 실현하는데 7.4.4의 Bader et al.의 알고리듬을 사용하시오. 원소 ＜D0,1＞의 자료 D0은 목적지마다 1로 전송되여야 한다는것을 가리킨다. 알고리듬의 4개 걸음들에서 매 마디의 내용을 보여 주는 그림들을 순서대로 그리시오. H의 값은 얼마인가?

마디 0	마디 1	마디 2	마디 3
<D0,0>	<D4,1>	< D7,2 >	<D10,3>
<D1,2>	<D5,2>	<D8,2>	<D11,2>
<D2,3>	<D6,3>	<D9,3>	<D12,3>

제3편. 체계구성방식

최근 년간 확대가능병렬체계들의 연구와 개발에서는 흥미 있는 문제들이 나타나고 있다. 더 많은 새로운 제품들이 늘어 나고 이 령역에서 전문적인 협의들이 널리 진행되고 있는것은 중요한 특징의 하나이다.

대학들에서 만든 일부 연구견본기계들이 공업생산에 이전되고 있다. 확대가능한 업무응용들과 클라스터가동환경들은 급속히 늘어 나고 있다.

시장요구와 응용경향은 지난 시기 체계구조개발을 추동해 온 2개의 주요요인들이다.

시장요구

Compaq로부터 IBM에 이르는 거의 모든 주요콤퓨터제작업체들은 병렬 또는 클라스터체계제품들을 밀어 내기 위하여 경쟁하고 있다.

Microsoft로부터 Oracle에 이르는 쏘프트웨어회사들은 《병렬준비된(parallel−ready)》체계들과 응용쏘프트웨어제품들을 발표하는데로 돌진하고 있다. 시장연구자들은 저규모에서 중규모까지의 병렬봉사기들에 대한 건전한 장성을 예측하고 있다.

정보기술경영자의 80프로가 지금 앞날의 전개계획으로 병렬체계를 생각하고 있다. 몇년전까지만 하여도 15%였다.

2000년에는 다음과 같은 2가지 형태의 콤퓨터들만이 존재할것이라는 합의도 커가고 있다.

- 탁상형, 무릎형, 노트형(set−to−box), 망(NC)기계들과 같은 여러가지 값 눅은 의뢰기기계들
- 이 편에서 취급하는 모든것을 포함하는 여러가지 확대가능봉사기들

응용경향

인터네트와 인트라네트, World Wide Web에서의 폭발적인 장성은 확대가능체계들의 연구와 개발을 더욱더 촉진하고 있다. 전세계적으로 수백만의 사용자들이 접속되면 그들은 단순한 전자우편봉사이상으로 요구할것이다.

인터네트전화, 텔레비죤회의, 매체봉사기들, 정보봉사기들, 전자상거래, 인터네트에 기초한 초고속계산모두는 확대가능하고 고도로 믿음직한 병렬봉사기들, 초고속봉사기들을 요구한다.

초고속봉사기에 대한 여러 응용동향들은 다음과 같다.

- 기술계산은 여전히 고도로 중시되는 한편 병렬체계의 응용들은 상업자료기지연산들(실례로 OLTP와 OLAP, 자료창고)과 망에 기초한 응용들에 의해 지배될것이다.
- 병렬기계사용에서 최근개념은 봉사기강화나 LAN강화이다. 여기서 대규모병렬

봉사기는 더 좋은 관리가능성과 확대가능성을 제공하기 위하여 여러개의 분산된 소형봉사기들을 대신한다.

- 응용들과 체계의 쏘프트웨어개발은 하드웨어설비보다 더 중요하게 되고 있다.
- 확대가능성과 유용성, 관리가능성은 단일한 큰 문제해결에서의 속도증가보다 점차로 더 중요하게 된다.
- 대형콤퓨터들과 벡토르초고속콤퓨터들은 계속 존재하도록 하는데 전심하고 있으며 한편 SMP들과 CC-NUMA기계들, 클라스터들은 자료창고와 초고속계산의 응용들을 위한 봉사기시장의 중요몫을 가지고 있다.
- 하드웨어와 쏘프트웨어에서는 더욱더 상품구성요소들이 사용될것이다. 그러나 새로운 응용들과 기능이 중요한 부분체계들을 지원하기 위하여 혁신적인 구성요소들이 여전히 필요하다.
- 계산 및 I/O능력들은 더 높은 R/D효과를 요구하는 체계의 병목으로 되고 있다.

확대가능체계의 4가지 종류가 이 편에서 고찰된다. 일부 연구과제들이 포함되며 공업에서 대표적이고 혁신적인 특징을 가지는 상업체계들을 강조한다. 매 체계에 대하여 그 체계에 고유한 하드웨어구조와 체계쏘프트웨어, 특수한 특징들을 서술한다. 특별한 기계가 포기될 때에도 사용될수 있는 바탕기술에 특별한 주의를 돌린다.

8장 이 장은 SMP와 CC-UNMA체계들을 론의한다. 이 체계들은 수많은 현존응용들과 함께 하드웨어지원단일기억기공간을 가진다. SMP들은 오늘 봉사기시장을 형성하고 있다. CC-NUMA기계들은 SMP구조를 수백개의 처리기로 확장한다.

9장 이 장은 클라스터화의 원리를 포함하고 있다. 클라스터의 구조들과 설계문제점들로부터 시작한다. 유용성과 단일체계영상을 위한 하드웨어 및 쏘프트웨어지원을 론의한다. 마지막에 클라스터일감관리의 문제들을 론의한다.

10장 이 장은 웅대한 연구과정으로부터 시작하는 여러가지 클라스터체계들과 상업클라스터들을 서술한다. 다음 Berkelay NOW와 IBM SP2, Digital TruCluster을 상세하게 론의한다.

11장 이 장은 여러개의 MPP구조들을 제기한다. MPP기계들은 값이 비싸지만 가장 좋은 크기확대가능성을 가진다. 거대한 응용들은 여전히 수천개의 처리기들로 된 체계에서 Tflop/s속도를 요구한다.

8장은 SMP와 CC-UNMA체계들이 아직도 병렬시장을 지배하기때문에 봉사기설계자들에게는 결정적으로 중요하다. 2개의 클라스터장들은 9장, 10장의 순서로 읽어야 한다. SMP와 NUMA, 클라스터들은 병렬 및 분산콤퓨터의 미래이다. 11장의 MPP들은 벡토르다중처리기들이 제한된 확대가능성을 가지므로 초고속콤퓨터공업의 재생을 암시하고 있다.

3편의 4개 장들은 콤퓨터설계자들과 체계설계자들, 응용프로그람작성자들이 반드시 읽어야 할 목록에 대한것이다. 이것들은 4편의 프로그람작성장들을 읽으려는 독자들에게는 필수적인것이라고 생각한다. 그러나 구성하는 기계들의 프로그람을 작성할 시간이 없는 하드웨어공학자들은 3편을 보지 않아도 된다.

제8장. 대칭 및 CC-NUMA 다중처리기

이 장에서는 확대가능한 공유기억기다중처리기들의 구조를 고찰한다. 최근의 대칭다중처리기(symmetric multiprocessor, SMP)들과 일관성캐쉬비단일기억기접근(coherent-cache nonuniform memory-access, CC-NUMA)기계들에 대한 설계경험을 자세히 고찰한다.

특히 2가지 SMP체계들인 Intel SHV체계기판과 Sun Ultra E-10000봉사기, 세가지 CC-NUMA체계들인 HP/Convex Exemplar X-class와 SGI/Cray Origin 2000, Sequent UNMA-Q 2000의 고유한 특징들을 고찰한다. 4개의 CC-NUMA다중처리기들에 대한 기술적비교는 마지막에 준다.

8. 1. SMP와 CC - NUMA기술

SMP구조는 오늘 사용되고 있는 거의 모든 병렬봉사기들에서 사용되고 있다. NUMA기계는 SMP체계의 확장이다. 아래에 구조적특징들과 SMP체계들에 대한 개괄, SMP체계에 대한 2가지 실례연구를 준다. NUMA기계들은 다음부분들에서 론의된다.

8. 1. 1. 다중처리기의 구성방식

공유기억기체계들의 공통구조를 그림 8-1 ㄱ)에서 보여 준다.

공유기억기체계(SMP 또는 CC-NUMA)들은 아래에 서술하는 여러가지 우월성을 가진다.

- **대칭성** 정규적인 조작체계보호를 제외하고 임의의 처리기는 임의의 기억기위치와 임의의 I/O장치에 접근할수 있다.
- **단일주소공간** 이 특징은 여러가지 리득을 가져 온다. 공유기억기에 상주하는 조작체계, 자료기지, 응용 등에 대한 오직 하나의 복사만이 있기때문에 단일체계영상은 자연히 지연된다. 단일OS는 프로쎄스들이 자기의 작업부하에 따라서 여러 처리기들우에서 실행하도록 일정을 짜므로 동적부하평형을 쉽게 달성한다. 그러므로 같은 공유기억기공간에 상주하는 모든 자료에 대하여 사용자들은 자료분배와 재분배를 걱정하지 않아도 된다.
- **캐쉬화** 자료위치는 캐쉬들의 계층으로 지원된다.
- **일관성** 캐쉬일관성은 하드웨어에 의하여 실시된다. 그러나 서로 다른 SMP들이나 CC-NUMA기계들은 서로 다른 공유기억기모형을 지원할수 있으므로 이것들중 일부는 일관성을 담보하기 위하여 프로그람작성자들의 조정을 요구할수 있다.
- **기억기통신** 공유기억기통신은 그것이 간단한 load/store지령들에 의해 진행되고 하드웨어가 일관성정보를 생성하므로 낮은 지연을 가진다. 이것은 많은 지령들을 실행하여야 하며 캐쉬일관성정보를 리용하지 않는 다중콤퓨터(6.6을 참고)의 I/O통신과 현저한 차이가 있다.

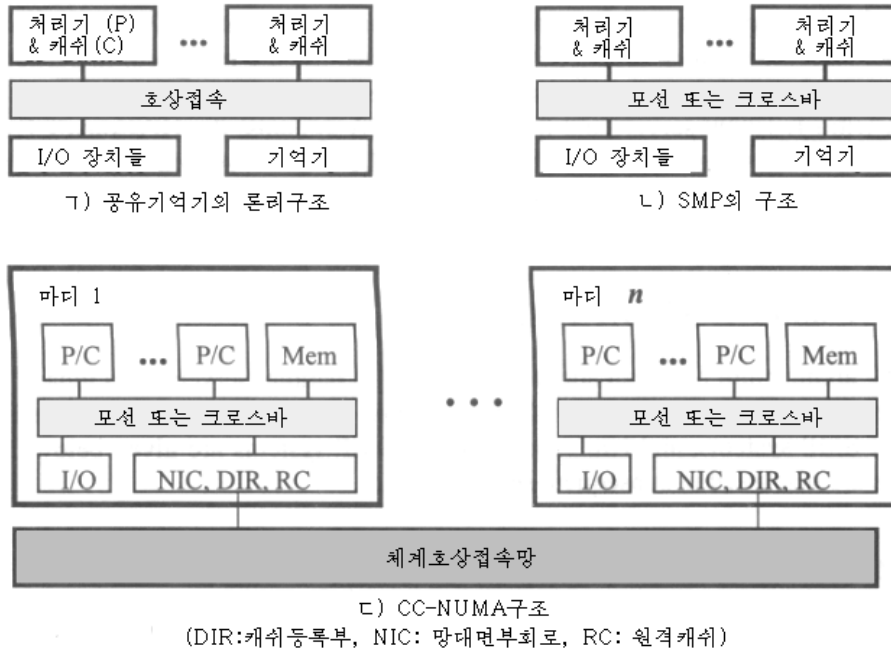



그림 8-1. SMP와 CC-NUMA체계들의 공통구조

대칭다중처리기

대부분의 상업적인 SMP체계들은 그림 8-1 ㄴ)에서 보여 주는 모선 또는 크로스바를 가정한다. 대부분의 SMP들은 모선호상접속을 사용하며 캐쉬일관성은 어떤 MESI와 같은 snoopy규약을 통하여 실시된다.

모선접속SMP들은 상업적으로 성공하였으며 오늘 병렬콤퓨터시장의 큰 몫을 가지게 되였다. 아래에 서술한것은 SMP들이 갖추어야 할 기본문제들이다.

- **유용성** 이것은 대체로 임의의 SMP의 가장 큰 문제이다. 모선이나 기억기, 조작체계의 실패는 전체 체계를 파괴한다.
- **병목** 모든 처리기들과 I/O조종기들은 기억기모선과 공유기억기를 놓고 경쟁하는데 이것은 병목으로 된다. 이 문제를 완화하기 위하여 대부분의 SMP들은 분산트랜잭션(파케트교환식모선)과 다중비차단특수요청과 같은 기술을 리용하고 있다.
- **지연** 다중콤퓨터들에 비하여 SMP들은 작은 지연을 가진다. 그러나 처리기의 속도에 비하면 아직 지연이 크다. 경쟁은 지연을 더욱더 길게 할수 있다. 경쟁을 무시한다 해도 현존 SMP의 최소기억기지연은 흔히 수백처리기주기이며 수천개의 지령들을 실행할 때는 훨씬 더 길다. 지연의 감소는 처리기속도에서의 증가와 보조를 맞추지 않았다.
- **기억기대역너비** 기억기대역너비는 처리기속도 또는 기억기용량의 증가와 보조를 맞추지 않았다. SGI의 John Mashey는 기억기와 디스크용량은 3년에 4배씩 증가하고 SMP의 기억기모선대역너비는 3년에 2배씩만 증가한다고 평가하였다.
- **I/O대역너비** 개별적인 I/O모선대역너비의 증가속도는 더욱더 낮다. 체계모선대역

너비를 증가시키는 한가지 방법은 그것을 크로스바교환기망으로 교체하며 모든 처리기들을 모든 기억기와 I/O장치들에 접속하는것이다.

- **확대가능성** 모선은 확대가능하지 않으며(6.6.1을 참고) 처리기들의 최대수를 수십개까지로 제한하고 있다. 많은 응용환경에서 공유된 모선/기억기우에서의 경쟁으로 하여 4개이상의 처리기들은 유력하게 사용될수 없다.
- 더 큰 체계에로 확장하기 위하여 3가지 방법 즉 모선/크로스바호상접속을 사용하는 방법(실례로 8.1.2에서 론의한 Sun Ultra-10000), CC-NUMA구조를 사용하는 방법(이 장의 뒤에서 론의한다.), 클라스터화를 사용하는 방법(9장과 10장에서 론의된다.)이 리용되였다.

기억기지연

SGI의 Larry McVoy에 따라 여러가지 형태의 기억기읽기지연이 있다.

첫번째는 읽기요청이 처리기소편을 떠나는 시간부터 자료가 기억기로부터 처리기소편에로 돌아 오는 시간까지로 측정한다. 이것은 본질적으로 기억기읽기모선주기의 시간이다. 이것은 모두 몇모선박자에서 수십모선박자까지의 범위에 있다. 실례로 Intel SHV에서 기억기읽기모선주기는 13모선박자이다. 66MHz(15ns)로 동작하는 모선에서 그것은 195ns이다.

두번째 지연의 정의는 페지화부가처리나 모선에서의 경쟁, 이전의 특수요청으로부터의 간섭이 없다는 가정하에서 처리기가 기억기접근동작을 시작하는 시간부터 자료가 적당한 등록기로 들어 가는 시간까지로 측정한다. 이것을 **최소기억기지연**이라고 한다. 이 지연은 기억기읽기모선주기의 여러배일수 있다.

세번째 지연정의는 경쟁과 간섭의 부가처리들이 포함된다는것을 제외하고 두번째 정의와 같다.

실례 8.1. SMP체계의 유효기억기대역너비

1997년에 전형적인 SMP체계에서 사용된 모선은 66MHz근방의 박자로 동작한다. 몇개만이 100MHz를 릉가한다.

F=200MHz로 동작하는 kB폭의 강화된 모선을 사용한다고 하자. 모선트랜잭션은 주소매김에 1박자, 중재에 1박자, 캐쉬일관성에 1박자, mB의 블로크자료를 넘기기하는데 [m/k]박자를 요구한다. $k=64, b=8, m=16B$일 때 제조업자가 공개한 자료는 모선대역너비 $k\times f=8\times100=80$MB/s를 요구한다. 그러나 유효대역너비는 최대로 $(m/3+[m/k])\times f=(16/(3+16/8))\times100=320$MB/s 이다.

지금 제조업자는 모선폭을 255b 또는 32B로 증가하고 32×100=3200MB/s의 대역너비를 요구한다. 유효대역너비도 4배 증가하겠는가?

대답은 《아니》이다.

최대유효대역너비는 단지 (16/(3+16/32))×100=400MB/s 또는 25% 증가된다. 물론 3박자의 부가처리는 더 큰 블로크크기 *m*을 사용함으로써 삭감될수 있다. 실험에서 *m*은 흔히 캐쉬행크기의 웃한계를 가지며 많은 고급(high-end)체계들에서 32B부터 256B근방에 있다.

행크기 m이 64B까지 확대되면 유효대역너비는 (64/(3+64/32))×100=128MB/s로 되며 여전히 32×100=3.2GB/s의 하드웨어최대값보다 훨씬 작다. 그렇게 계산한 유효대역너비는 달성가능한 최대값이다.

CC−NUMA체계

그림 8−1 ㄷ)에서 보여 준바와 같이 CC−NUMA기계는 SMP마디들을 더 큰 체계에 접속함으로써 SMP들을 확장한다. 대부분의 CC−NUMA다중처리기체계들은 등록부에 기초한 캐쉬일관성규약을 사용한다.

CC−NUMA기계는 SMP구조의 우월성을 유지하는 한편 관습적인 SMP들의 확대가능성문제를 완화한다. 분산된 공유기억기구조는 확대가능성을 높인다. 더 많은 마디들을 첨가함으로써 처리기들뿐아니라 기억기용량과 I/O능력을 늘일수 있다. 경쟁과 대역너비의 문제들은 응용이 대부분시간에 자료국부성의 우월성을 가지고 여러 국부기억기들에 동시에 접근할수 있기때문에 완화된다.

일부 CC−NUMA기계들은 지어 SMP의 유용성문제도 풀려고 시도하고 있다.

실례로 SGI Cellular IRIX조작체계는 세포의 개념을 실현하므로 조작체계의 한 부분에 대한 여러개의 복사들을 여러 마디들우에서 실행할수 있고 한 마디의 실패가 전체 체계를 파괴하지 않는다.

다중콤퓨터들이나 NCC−NUMA기계들에 비한 CC−NUMA의 주목할만한 우월성은 프로그람작성자가 자료구조들을 마디들에로 명백히 분배하지 않아도 된다는것이다. 체계의 하드웨어와 쏘프트웨어는 처음에 자료를 마디들에 자동적으로 분배한다. 응용의 실행시간동안 캐쉬일관성하드웨어는 자료를 그것들이 참조되는 마디들에로 자동적으로 이동시킨다. 그러나 이것은 다음의 실례에서 밝혀 지는바와 같이 언제나 유효하게 실현되는것은 아니다.

실례 8.2. CC−NUMA체계의 원격캐쉬화문제

하나의 프로그람이 자료배렬 A와 B에 접근하는 다음의 코드를 실행하는 2개의 프로세스 P와 Q를 가진다고 가정하자.

	P:	Q:
단계1:	MSE(A)	MSE(B)
단계2:	MSE(B)	MSE(A)

체계는 처음에 자료를 그림 8−2 ㄱ)와 같이 배정하는데 이것은 단계 1에 대하여 리상적이다. 통신요구는 없고 원격캐쉬들은 비여 있다. 원격캐쉬는 그것들의 저장장소가 원격기억기안에 있는 자료를 꺼내는데 사용된다는데로부터 이름지어 졌다.

단계 2에서 하드웨어는 그림 8−2 ㄴ)와 같이 B를 마디 1로, A를 마디 2로 자동적으로 옮긴다.

프로그람작성자는 다중콤퓨터에서 요구되는 자료재분배를 하지 않아도 된다.

이제는 문제로 돌아 가자.

원격캐쉬는 마디안의 국부기억기보다 훨씬 더 작다.

실례로 Sequent CC-NUMA-Q에서 원격캐쉬는 32MB이고 한편 국부기억기는 4GB이다. 단계 2에서 프로그람의 작업설정이 원격캐쉬보다 훨씬 더 크다면 어떻게 되겠는가?

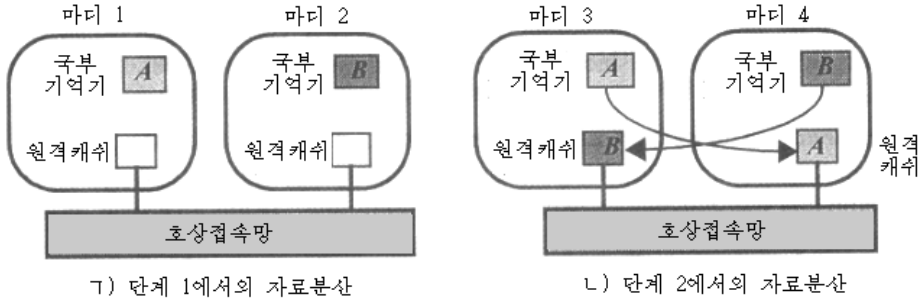



그림 8-2. CC-NUMA다중처리기체계의 원격캐쉬사용에서 자료재배치

결과는 용량불일치가 일어 나 원격캐쉬들의 내용들을 원격마디안의 본래의 기억기로 돌아 와 써야 한다. 이 원격캐쉬문제를 풀기 위하여 CC-NUMA기계안의 조작체계들은 여러 기술들을 통합하고 있다.

실례로 동족일정짜기(affinity scheduling)기술은 단계 2에서 프로쎄스 P는 그것의 자료에 가까운 마디 2에서 실행하도록 하는 한편 프로쎄스 Q는 마디 1에서 실행하도록 옮겨 진다. 페지이동기술은 자료 B의 페지를 마디 1의 국부기억으로 옮기고 자료 A의 페지를 마디 2의 국부기억으로 옮긴다.

CC-NUMA체계의 하드웨어와 쏘프트웨어에서 혁신적인 기술들은 자료의 국부성을 개척하고 확대가능성을 높이는데서 대단히 유효하다.

상업응용망들에서 자료접근의 대부분은 국부마디내에서 충족되며 마디에서의 대다수통신들은 자료전송때문인것이 아니라 캐쉬무효때문이라는것을 보여 주는 일부 증거가 있다.

8. 1. 2. 상업SMP봉사기

지금 SMP봉사기들은 사용에서 가장 성공적인 병렬콤퓨터로 되고있다.

표 8-1에서 5개의 체계를 개괄한다. 다음 2개의 전형적인 SMP체계들을 상세히 론의한다. Intel SHV의 4개 처리기봉사기는 낮은급을 대표하며 많은 NUMA체계들에서 구성요소로 널리 사용되였다.

Sun Ultra Enterprise 10000는 고급을 대표하며 64개 처리기로까지 확대가능한데 8.2에서 취급된다.

표 8-1의 파라메터들은 모두 최대값 또는 가장 좋은 값이다.

일부 특징들은 모든 SMP들에서 공통적인데 아래에서 설명한다.

- **성능** 모든 SMP들과 더 많고 더 빠른 처리기들뿐아니라 더 큰 캐쉬/기억기/디스크 용량, 더 많은 I/O확장홈들, 더 높은 기억기 및 I/O대역너비, 더 낮은 기억기접근 지연을 포함하는 확대가능성능을 제공하기 위하여 노력하고 있다.
- **회복력** 체계는 구성요소의 고장을 곧 회복할수 있어야 하며 응용의 가동시간을 최대화하여야 한다. 알려 진 기술들은 믿음직하고 여유가 있는 교체가능한 구성요소들(전원공급을 포함)을 사용하는것, 예보적인 진단과 체계감시, 검사를 수행하는 봉사처리기를 사용하는것, 여러 SMP들이 더 높은 유용성을 제공하기 위하여 쉽게 클라스터화될수 있는 클라스터준비된 체계를 만드는것, 광범한 검사를 하는것과 자료기지 및 응용들과 통합하는것을 포함한다.
- **통합** 체계는 혼합된 판매자환경내에서 응용들과 미들웨어, 조작체계들, 망들의 통합을 헐하게 하는 호상운영성을 제공하여야 한다. 리상적인 봉사기는 대형콤퓨터와 Unix, Windows NT, 다른 가동환경들과 흠없이 통합할수 있어야 한다.
- **보안** 많은 SMP봉사기들은 <분리된 대형콤퓨터+말단들>의 형태대신에 망환경에서 동작하고 있다. 보안은 특히 인터네트와 인트라네트, 자료쎈터들의 대중화와 관련하여 중요하다.
- **관리** 체계는 구성과 고장들, 자원, 성능, 료금계산기능을 관리하는 능력을 제공하여야 한다.

8. 1. 3. Intel SHV봉사기기판

Intel의 표준고볼륨(standard high-volume, SHV)봉사기기판의 대표적인 형태를 그림 8-3에서 보여 준다. 그것은 4개의 Pentium Pro(P6)묶음들과 2개의 PCI다리, 1 또는 그이상의 기억기조종기들, 하나의 OEM다리로 구성된다.

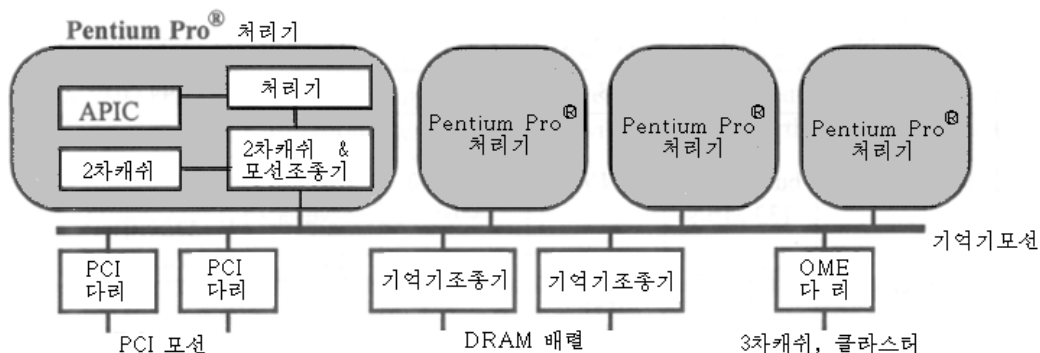



그림 8-3. Intel SHV SMP기판구조

이 구성요소들을 **모선발동자**(bμs agent)라고 하며 66MHz, 64bit기억기모선에 련결한다.

PCI다리들은 디스크들과 CD-ROM들, 다른 I/O장치들에로의 접속을 제공하는 2개의 PCI모선들을 기억기모선에 대면부로 련결시킨다.

기억기조종기들은 4GB까지의 DRAM들을 관리한다.

OEM다리는 원하는 무엇이나 거기(실례로 3차캐쉬)에 넣을수 있는 3부류(third-party)개발자들에 의해 구성되도록 공개되였다.

대중적인 사용은 여러 SHV기판들을 큰 클라스터나 CC-NUMA기계들에로 접속하는 교착론리이다.

표 8-1　　　5개의 상업 SMP체계들의 비교

체계특징	DEC Alphaserver 8400 5/440	HP 9000/T600	IBM RS 6000/R40	Sun Ultra Enterprise 6000	SGI Power Challenge XL
처리기의 수	12	12	8	30	36
처리기형태	437MHz Alpha 21164	180MHz PA8000	112MHz PowerPC 604	167MHz Ultra SPARC I	195MHz MIPS R10000
처리기당 소편외장캐쉬	4MB	8MB	1MB	512KB	4MB
호상접속대역너비	Bμs 2.1GB/s	Bμs 960MB/s	Bμs+Xbar 1.8GB/s	Bμs+Xbar 2.6GB/s	Bμs 1.2GB/s
I/O통로들	PCI모선 12개, 매개가 133MB/s	N/A	MCA 2개, 매개가 160MB/s	Sbμs 30개, 매개가 200MB/s	Power Channel-2 HIO 6개, 매개가 320MB/s
I/O확장홈들	PCI확장홈 144개	HP-PB확장홈 112개	15MCA	Sbμs 확장홈 45개	HIO확장홈 12개
I/O대역너비	1.2GB/s	1GB/s	320MB/s	2.6GB/s	HIO확장홈당 320MB

기억기모선

Intel SHV기억기모선은 많은 SMP모선들에 공통인 여러 특징들을 통합하고 있다. 그것은 SHV에서는 MESI규약인 하드웨어캐쉬일관성규약을 지원한다. 그것은 모선중계와 주소만들기, 요청, 응답과 같은 서로 다른 과제들을 수행하는데 따로따로의 신호선을 사용한다.

기억기접근들은 관흐름화될수 있다.

높고 지속적인 대역너비를 제공하기 위하여 분산트랜잭션과 특수한 요청들이 지원된다.

이것은 15ns 또는 66.7MHz의 모선박자를 가진다. 보통 그림 8-4에서 마지막특징을 가지는 SHV기억기모선은 **파케트교환식모선**이라고 부른다.

SHV기억기모선은 보여 주는바와 같이 관흐름된 순서 있는 트랜잭션들을 수행한다. 순서 있다는것은 모선요청들이 먼저 온것은 먼저 봉사하는 원리로 봉사 받는다는것을 의미한다.

모선의 신호선들은 6개의 개별적인 묶음(부분모선)들로 나눌수 있다.

기억기읽기모선주기는 요청부분모선을 획득하기 위하여 중계부분모선을 주장하는것으로 시작한다.

이것은 2박자 걸린다.

다음 요청이 두개의 린접하는 박자를 거쳐 요청부분모선우로 나온다. 첫번째 박자는 주소와 기억기형태를 포함하며 두번째 박자는 유일한 트랜잭션 ID, 요청길이 등을 포함한다. 요청후 3개 박자가 나온 다음 오유부분모선이 검사된다.

임의의 오유는 요청이 다시 나오게 한다.

요청후 4개 박자가 나온 다음 요청은 목적으로 한 모선발동자(agent)에 도착하며 요청이 응답을 요구하는가를 가리키도록 한다. 응답을 요구하지 않으면 기다림상태가 여러 개의 2개 박자안에 삽입된다.

발동자(agent)가 완료를 결정하면 경쟁부분모선을 주장하고 2개 박자후에 응답부분모선을 주장하여 자료를 자료부분모선우에 넣는다.

32B의 캐쉬행길이(cache line length)가 있다고 하면 자료를 넘기기하는데 4박자가 필요하다. 읽기모선주기는 총 13박자 걸린다.

6개의 부분모선을 관흐름단계들로 볼수 있다. 결과 모선트랜잭션은 13박자가 아니라 4박자마다 시작할수 있다.

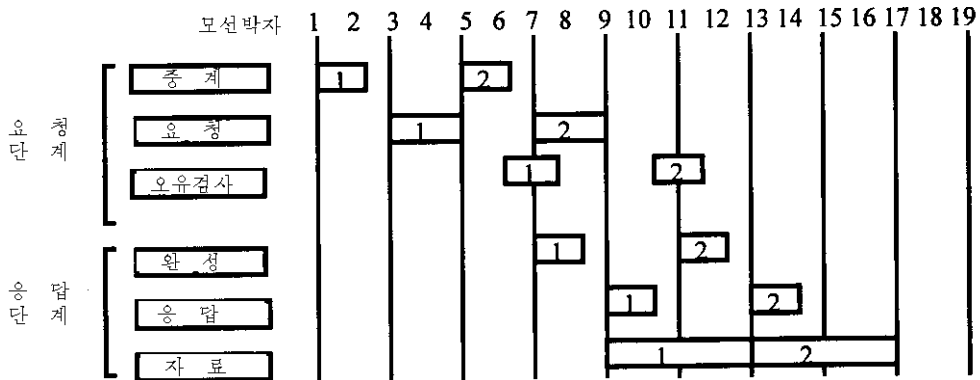



그림 8-4. 관흐름식Intel SHV모선읽기주기

모선은 여러 발동자들이 모선을 공유하도록 분산트랜잭션을 지원한다. 트랜잭션은 요청단계(중계,요청, 오유검사)와 응답단계(완성, 응답, 자료)로 분할된다. 모선은 트랜잭션의 요청단계를 실행할수 있으며 그다음 첫번째 트랜잭션의 응답단계를 실행하기전에 다른 트랜잭션을 실행할수 있다. 모선은 다른 트랜잭션을 실행하는 동안 4개까지의 특수한 요청들을 지원한다.

실례 8.3. Intel SHV SMP기판사용에서 분산트랜잭션

CC-NUMA기계가 구성요소로서 여러개의 SHV기판들을 사용한다고 하자. CC-NUMA기계의 설계자는 SCI고리로 SHV기판을 접속하기 위하여 OEM다리들을 사용한다.

2개의 읽기모선주기가 실행되는데 첫번째 읽기모선주기는 국부기억기에서 자료를 찾지 못한다고 하자. 그러면 원격기억기로 가야 하는데 이것은 130모선박자가 걸린다. 두번째 읽기모선주기는 항상 국부기억기에서 자료를 찾으며 단지 13박자 걸린다. 그러나 분산트랜잭션이 없으면 두번째 모선주기는 첫번째 모선주기가 끝날 때까지 기다려야 하며 이것은 그것의 지연이 130박자로 증가된다는것을 의미한다.

분산트랜잭션에서 첫번째 트랜잭션은 그것이 인차 끝날수 없다는것과 두번째 트랜잭션이 13박자에 끝나도록 모선을 양보할것을 모선에 알린다. 그동안 첫번째 트랜잭션의 특수한 요청은 OEM다리에서 유지되며 원격접근이 끝나기를 기다린다. 자료가 도착하면 OEM다리는 자료를 모선우에 넣고 트랜잭션을 끝낸다. 모든 트랜잭션이 유일한 ID를 가지므로 혼란은 없다.

8. 2. Sun Uitra Enterprise 10000체계

Ultra Enterprise 10000은 Sun의 고급SMP봉사기이며 StarFire라고도 부른다.
이 봉사기는 성능과 유용성을 높이기 위하여 여러 새로운 기술들을 통합하고 있다. 이것은 64개의 처리기, 64GB의 기억기, 20TB이상의 직렬디스크까지 확장할수 있다.

체계는 모든 Sloaris응용들을 실행하며 대형콤퓨터환경들을 통합하기 위한 접속성과 호상운영성의 지원을 제공한다.

8. 2. 1. Ultra E-10000의 구성방식

Enterprise 10000은 Enterprise 6000체계에 비하여 확대가능성과 믿음성, 유용성, 봉사성에서 커다란 구조적발전을 이룩하였다. Enterprise 10000의 블로크도를 그림 8-5에서 보여 준다.

체계는 아래에 서술하는바와 같이 높은 대역너비의 중심판(centerplane)주위에 구성된다.

Gigaplane_XB호상접속

Gigaplane_XB호상접속은 Enterprise x000봉사기들(실례 6.1)의 Gigaplane체계모선에 비하여 대역너비에서 10배 개선된다. Gigaplane_XB호상접속은 2개 수준의 크로스바에 기초하고 있다. 이 호상접속은 체계기판의 모든 기억기들과 I/O장치들을 하나의 단일한 주소공간으로 련결함으로써 임의의 처리기가 전체 체계의 임의의 기억기나 I/O장치에 접근할수 있게 한다.

호상접속은 Ultra포구구조(Ultra Port Architecture,UPA)표준을 고수한다. Ultra-1탁상용워크스테이션들과 Enterprise봉사기들을 위한 기본모선인 UPA모선은 CPU/기억기기판들과 I/O기판들을 Gigaplane모선에 접속하기 위한 중간모선으로 사용된다. UPA모선은 1.3GB/s의 최대대역너비를 가지고 8.3MHz로 동작한다.

체계기판에서 UPA는 4개의 처리기들과 I/O모듈이 기억기와 동시에 대화하게 하는 크로스바이다. UPA는 매 포구당 128b의 자료폭을 가진다. UPA는 또한 대역적인 Gigaplane크로스바에로 또는 대역적인 Gigaplane크로스바로부터의 자료파케트경로정하기도 담당한다.

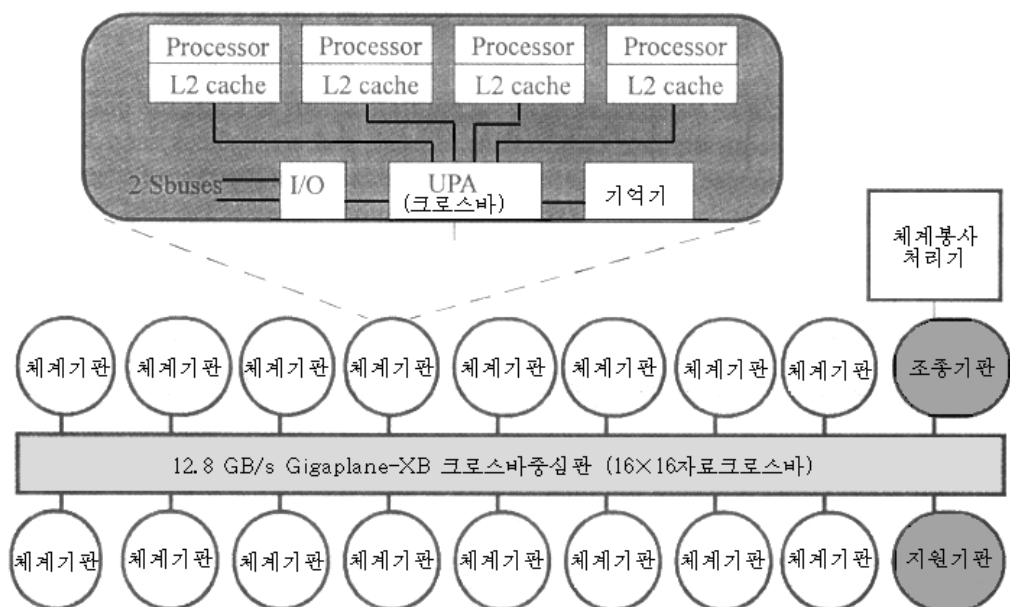



그림 8-5. Sun Ultra Enterprise 10000의 구조

물리적으로 Gigaplane-XB호상접속은 2개의 대칭면으로 된 회로기판의 중심판(centre-plane)으로 실현된다. 매면은 8개까지의 체계기판을 설치하며 중심판은 기판들과 조종기판을 지원한다. Gigaplane-XB는 16B폭의 16×16자료크로스바와 4개의 주소모선들, 2개의 전원분배모선들로 구성된다.

개선된 박자속도

Ultra SPARC-I처리기는 체계박자속도와 처리기박자속도가 1:2 또는 1:3이 될것을 요구한다. 초기의 E-10000체계는 83.3MHz의 체계박자와 250MHz의 처리기를 사용한다. 그러나 설계에서는 더 빠른 처리기들을 정합하도록 100MHz의 체계박자를 사용할수 있다.

4개의 주소모선

UPA는 분리된 주소와 호상접속을 정의한다. 모선식체계에서는 보통 선대역너비의 약 70%만이 자료에 사용가능하며 나머지는 주소와 조종에 사용된다.

2가지 기능의 분리는 리용가능한 대역너비의 더 좋은 리용을 가져 온다. Snoopy주소들은 모든 기판들에 동시에 방송되여야 하며 한편 자료파케트들은 크로스바교환기를 통하여 점대점으로 전송될수 있다.

4개의 대역주소모선들은 주소만들기요청을 모든 체계기판들에로 방송한다.

하나의 주소모선이 체계기판우의 4개의 기억기장소매개에 사용된다. 매 모선은 독립인데 이것은 4개의 명확한 주소전송이 동시에 있을수 있다는것을 의미한다. 주소전송은 2박자주기가 걸린다.

대역16×16자료크로스바

크로스바안에 확립된 16개의 자료경로가 있는데 매 자료경로는 16B 또는 256b폭이고 매개 체계기판에로 각각 접속할수 있다.

Enterprise 10000체계는 자료대역너비와 정합되는 충분한 주소대역너비를 공급하기 위하여 4개의 snoopy경로를 사용한다.

자료크로스바는 16개의 체계기판들사이에서 자료파케트들을 경로로 정한다. 자료는 64B의 파케트들로 편성되며 한개의 파케트를 전송하는데 4개의 체계박자가 걸린다. 16×16크로스바로 하여 Giagplane은 12.8GB/s만한 높은 집체대역너비를 제공한다.

Enterprise 10000체계는 물리기판의 분할에 기초한 2단경로정하기형태를 가진다. 국부적인 n대 1 경로들은 내장기판의 요청들을 모아서 하나의 외장기판의 포구에 접속한다. 대역자료크로스바는 매 기판의 하나의 포구를 한데 합쳐서 접속한다.

8. 2. 2. 체계기판의 구성방식

체계는 Gigaplane_XB중심판에 끼운 하나의 조종기판과 하나의 지원기판, 16개까지의 체계기판을 포함한다. 조종기판은 체계박자발생기와 온도/공기흐름감시와 같은 모든 체계기판들에 사용되는 체계수준의 론리를 포함한다. 조종기판은 이써네트를 통하여 체계봉사처리기(SSP, 체계 console)에 련결한다. 그것은 초기적재와 닫기, 감시, 진단, 그밖의 체계관리과제들을 처리한다.

체계기판

매 체계기판은 UPA크로스바로 호상접속된 4개의 처리기모듈과 한개의 기억기모듈, 하나의 I/O모듈을 포함한다. 체계기판의 구조를 그림 8−6에서 보여 준다.

모듈당 4GB까지의 용량을 가지는 기억기모듈은 64Mb ECC DRAM소편들로 된 4개의 끼워넣기식저장소를 포함한다.

I/O모듈은 2개의 표준 Sbμs(IEEE 1496−1993)사이에 다리를 놓으며 매 Sbus는 망화와 I/O를 위한 2개의 확장홈을 가진다.

Enterprise 10000은 자체의 I/O체계에 가상주소들을 사용하며 Sbus대면부는 가상 대 물리주소변환을 위한 자체의 기억기관리단위를 포함한다. 처리기의 내부박자는 250MHz이며 나머지체계기판은 83.3MHz(12ns)의 체계박자로 동작한다. 기억기모듈은 매 4개 체계박자(48ns)당 1개의 64B캐쉬행읽기 또는 쓰기를 수행할수 있으며 1.3GB/s의 대역너비를 제공한다. 64−b Sbus는 25MHz로 동작하며 100MB/s의 대역너비를 유지한다.

자료경로정하기

Enterprise 10000체계에서 자료경로정하기는 2개의 수준 즉 대역과 국부수준에서 처리된다. 대역자료경로조종기는 16개의 체계기판들사이에서 자료파케트들을 조종하는 18B폭의 16×16크로스바이다. 16×16크로스바로 하여 임의의 포구는 중심판전면에 걸쳐 임의의 다른것에 접속할수 있다. 18B중 16B는 자료용이고 나머지 2B는 오유수정을 위한것이다.

Star Fire의 주소경로정하기는 4개의 대역주소모선들의 분리된 모임우에서 실현된다. 주

소를 나르는 주소모선들은 방송되지만 실현은 실제상 점대점경로조종기이다. 그 의의는 경로조종기들이 모선보다 더 큰 내재적인 믿음성을 가진다는것이다. 모선들은 오유정정코드비트들을 포함하여 48b이다. 매 모선은 독립인데 이것은 4개의 명확한 주소전송이 동시에 있을수 있다는것을 의미한다.

주소전송은 2개 박자주기가 걸리며 주소모선우에서 초당 167×1000000 snoops의 snoopy 속도와 동등하다. 주소모선에서 고장이 생기면 새치기된 연산은 남은 모선들을 사용할수 있다.

8. 2. 3. 확대가능성과 유용성지원

StarFire체계에서의 확대가능성과 유용성을 높이기 위한 특수한 하드웨어 및 쏘프트웨어지원은 아래와 같다.

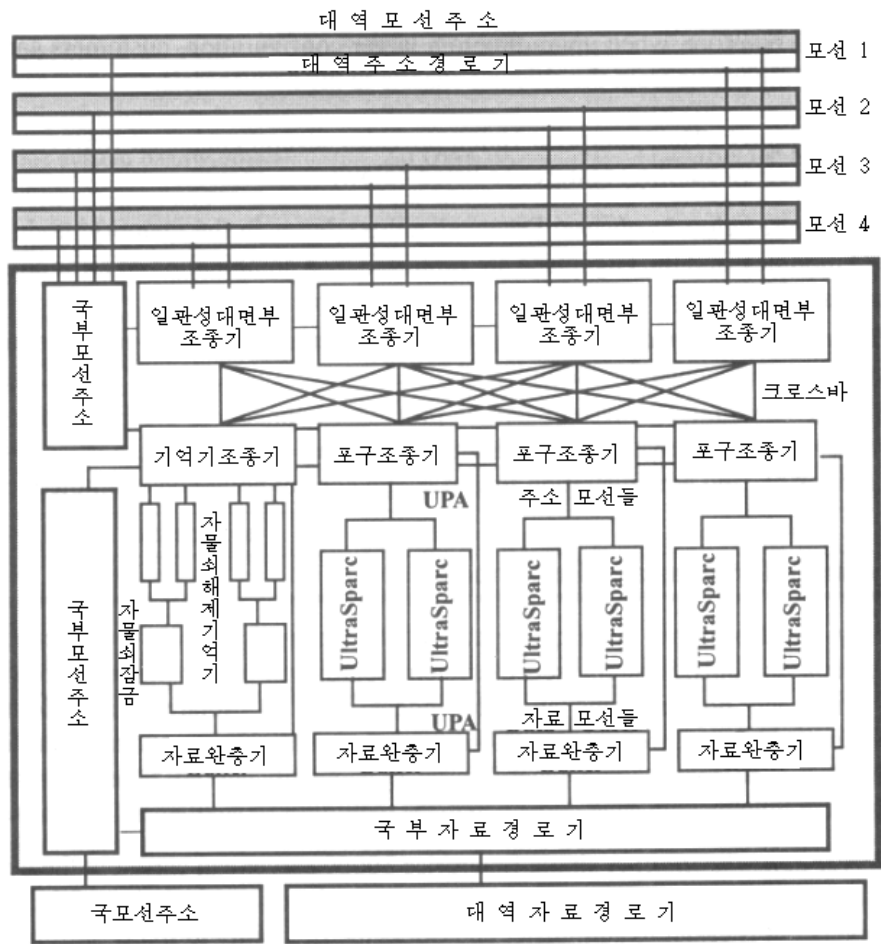



그림 8-6. Sun StarFire봉사기의 체계기판구조

확대가능성지원

강력한 워크스테이션들과 망화된 개인용콤퓨터들의 광범한 사용은 더 큰 SMP봉사기들을 요구한다.

Sun은 확대가능한 SMP개발에서 다음의 우월성들을 실현하였다.

- Gigaplane_XB호상접속과 캐쉬일관성기구는 Sun SMP봉사기들의 확대가능성을 높이는데서 결정적인 역할을 놀았다. 다른 중요한 역할은 Sun의 Solaris조작체계가 놀았다.
- StarFire는 16~64개의 CPU와 2~64GB의 기억기, 20TB이상의 직결디스크기억으로 구성될수 있다. 처리기들과 기억기, I/O들사이의 확장홈절충은 없다.
- Enterprise 3000, 4000, 5000, 6000봉사기들에서 사용된 모든 250MHz처리기모듈들과 SIMM들, Sbμs기판들은 Enterprise 10000봉사기에도 공통적이다. 그러므로 더 큰 구성으로 확대할 때 사용자들은 이 구성요소들을 현존 종류로부터 새로운 종류로 바꿀수 있다.

유용성지원

Enterprise 10000은 아래와 같은 여러개의 개선된 RAS(reliability, availability, serviceability)특성들을 제공한다.

- **오유정정하드웨어(Error−correction hardware)** 기억기모듈들과 I/O모듈들, Gigaplane안의 자료 및 주소모선들은 오유정정코드들과 기우성비트의 결합에 의해 보호된다.
- **즉시교체가능한 여유구성요소** 실례로 전원공급과 대부분의 기판준위의 구성요소들
- **봉사 console** 체계봉사처리기는 전통적인 이써네트를 통하여 Enterprise 10000봉사기에 접속하며 체계진단과 검사, 감시, 단일조종점과 원격위치로부터의 관리를 가능하게 한다.

8. 2. 4. 동적령역과 성능

동적체계령역

Enterprise 10000의 고유한 특징은 동적체계령역의 개념이다. 이것으로 하여 체계기판들의 모임들이 여러개의 령역으로 구획지어 질수 있으며 매 령역은 Solaris의 독립적인 복사를 실행한다. 매 령역은 하드웨어 또는 다른 령역에서 일어나는 쏘프트웨어오유로부터 완전히 분리된다. 령역은 실행시에 동적으로 형성되고 재구성될수 있다.

Sun은 동적체계령역들이 더 작은 개별적인 봉사기들의 모임에 비하여 여러가지 우월성을 가진다고 보고 있다.

- **봉사기통합** 단일한 Enterprise 10000봉사기는 여러개의 더 작은 봉사기들의 임무들을 수행할수 있다. 그것은 따로따로 떨어 져 있는 봉사기들의 독립성과 분리를 유

지하지만 한 봉사기에서 다른 봉사기로 자유롭게 옮기는 융통성을 관리하고 제공하기가 더 쉽다.

- **동시적인 개발과 생산, 검사** 이 3가지 기능들은 단일 Enterprise 10000체계내에서 안전하게 공존할수 있다. 령역들의 분리는 개발과 검사작업을 생산과 함께 계속할수 있게 한다.
- **쏘프트웨어이동** 이전의 SMP체계들에서 요구하는것과 같이 전체 체계가 행(line)을 떠나지 않고 한 령역에서 다른 령역으로 회전하는 형식으로 체계들이나 응용들의 쏘프트웨어를 갱신할수 있다.

성능비교

Sun Fire Enterprise 6000과 StarFire Enterprise 10000의 성능을 표 8-2에서 비교한다.

표 8-2 2개의 Sun SMP봉사기의 성능비교

성능형태	Enterprise 6000 (Sun Fire체계)	Enterprise 10000 (StarFire)
체계모선속도	2.5GB/s	83.3MHz에서 10.4GB/s
최대체계대역너비	2.6GB/s	100MHz에서 12.2GB/s
SPECrate-int95 SPECrate-Pp95	30개 CPU에서 2317 30개 CPU에서 1782	16~64개 CPU에서 N/A
체계모선처리량	2.5GB/s	10.4GB/s
기억기지연	300ns	500ns
망대역너비	622MB/s까지	622MB/s까지
I/O성능	1~30개 Sbμs, 200MB/s	2-32Sbμs, 200MB/s
	10개까지의 독립인 Sbμs	32개까지의 독립인 Sbμs
SCI성능	20MB/s	20MB/s
UltraSCSI성능	40MB/s	40MB/s
빛섬유통로	25MB/s	25MB/s

두 체계는 처리기당 1MB의 외장캐쉬를 가지는 250MHz Ultra SPARC Ⅱ처리기들을 사용하며 Solaris 2.5.1조작체계로 동작한다.

성능개선은 구조적특징에서 즉 30개의 CPU로부터 64개의 CPU로, 30GB의 최대기억기로부터 64GB의 최대기억기로, 45개의 Sbμs확장홈으로부터 64개의 Sbμs확장홈으로, 162GB의 내부디스크로부터 190GB의 내부디스크로, 10TB이상의 전체 디스크로부터 20TB이상의 전체 디스크로의 상승을 가져 온다.

체계호상접속에서의 개선은 StarFire의 확대가능성을 SunFire이상으로 높이였다.

크게 보충된 RAS특징들은 StarFire의 유용성을 SunFire이상으로 높이였다.

기억기지연의 증가는 SunFire의 구성과 기억기용량을 더욱 확대하였기때문에 생긴다.

8. 3. HP/Convex Exemplar X-Class

Exemplar는 1996년에 Hewlett-Packard/Convex이 확대가능병렬처리기(SPP)로서 소개한 CC-NUMA기계이다. 이 기계는 물리적으로 분산된 기억기모듈들과 캐쉬들에 대하여 캐쉬/기억기의 일관성을 위한 하드웨어지원을 사용하여 단일공유기억기공간을 제공한다.

Exemplar는 Convex Exemplar SPP 1600체계의 계승체계이다. 이것은 기술계산과 다량의 자료기억관리, 망봉사기 등을 위한 일반목적가동환경으로서 설계된다. 구조는 512개의 처리기들과 512GB의 물리기억기까지 확장할수 있다.

8. 3. 1. Exemplar X System의 구성방식

Exemplar X의 구조를 그림 8-7에서 보여 준다.

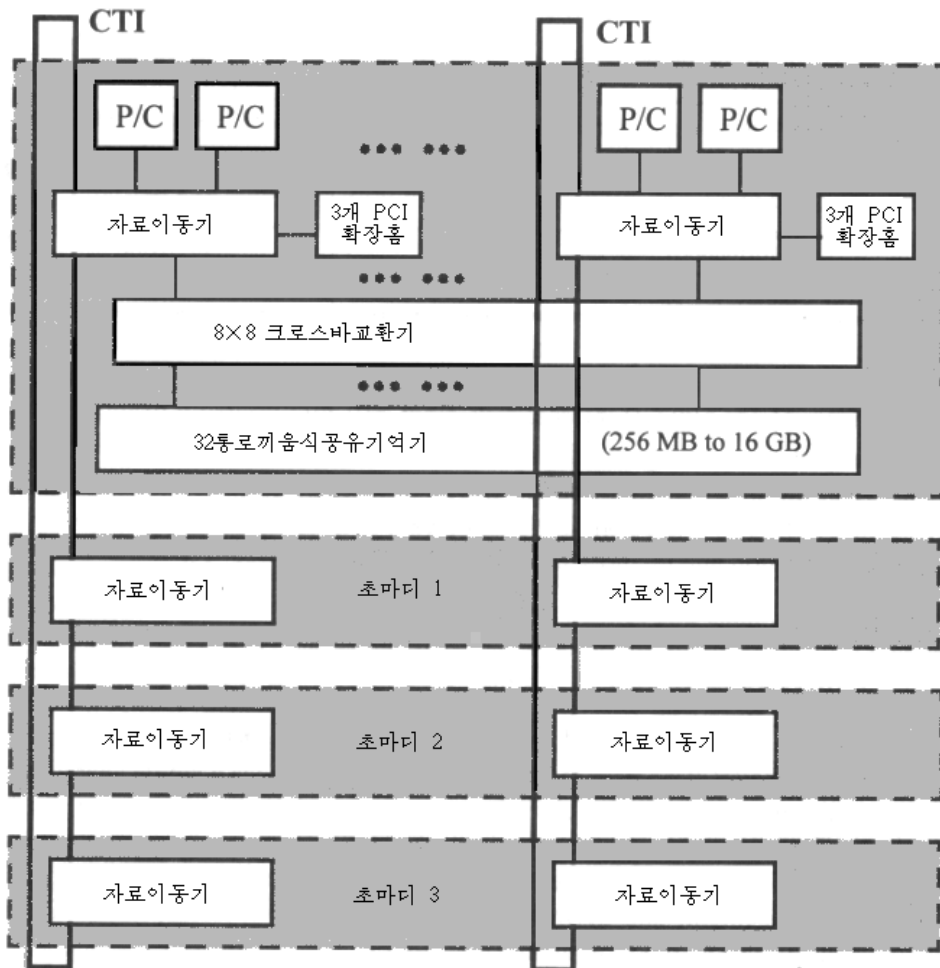



그림 8-7. HP/Convex Exemplar X-Class SPP의 구조

Exemplar X체계는 일관성고리면호상접속(Coherent Toroidal Interconnect, CTI)을 통하여 접속된 여러개의 초마디들(hyper−node)로 구성된다. 매 초마디는 180MHz박자의 64b PA−8000처리기들을 16개까지 가지는 SMP체계(HP/Convex에 의해 S−Class라고 부른다.)인데 매 처리기는 1MB의 지령캐쉬와 1MB의 자료캐쉬를 가진다.

처리기/캐쉬는 그림 8−7에서 P/C로 표시된다.

체계하드웨어와 쏘프트웨어는 임의의 처리기와 임의의 I/O장치조종기(실례로 DMA엔진)가 전체 체계에서 서로 다른 초마디에 포함되는 임의의 기억기에 접근하게 하는 대역적으로 공유된 기억기를 제공한다.

류사하게 임의의 처리기는 임의의 초마디에 련결된 임의의 장치에 접근할수 있다. 하드웨어는 체계폭캐쉬일관성을 유지한다.

물리적으로 같은 주소공간에 4개 수준의 계층기억기가 있다.

처리기는 대부분시간동안 자기의 캐쉬들에만 접근하여야 한다.

캐쉬실패의 경우 처리기는 크로스바교환기를 통하여 같은 초마디의 국부공유기억기에 접근한다. 자료가 국부기억기에서 발견되지 않으면 처리기는 CTI를 통하여 다른 초마디의 기억기로부터 자료를 얻는다.

원격기억기접근을 줄이기 위하여 매 초마디는 이전에 다른 초마디들이 접근한 자료들을 꺼내는 CTI캐쉬(클라스터캐쉬로도 알려 져 있다.)를 포함한다. CTI캐쉬는 물리적으로 지적되며 대역물리주소가 붙여 진다.

Exemplar X−Class체계의 기본파라메터들을 표 8−4에서 개괄한다.

CTI호상접속은 SCI표준으로부터 파생된다. 이것은 초마디의 모든 공유기억기들에 대한 체계폭일관성접근을 제공한다. CTI는 점대점 한방향련결들의 렬로서 실현된다. SCI와 같이 CTI는 여러개의 특수한 요청을 허용하는 분산트랜잭션규약을 사용한다.

하나의 원천지초마디가 하나의 목적지초마디로부터 캐쉬행을 요청한다고 하자.

표 8−3　　Exemplar X−Class의 자원 및 성능속성들

속성	초마디	X−Class체계
처리기의 수	4~16	16~64
최대속도(Gflop/s)	11.5	46
캐쉬기억기용량(MB)	8~2	32~128
물리기억기용량(GB)	0.256~16	1~64
최대기억기대역너비(GB/s)	15.38	61
PCI조종기의 수	1~24	1~96
내부디스크용량(GB)	148	592
최대I/O대역너비(GB/s)	1.92	7.68
최대CTI련결대역너비(MB/s)	−	450

원천초마디하드웨어는 CTI고리를 거쳐서 요청파케트를 내보낸다. 요청파케트는 고리를 거쳐서 목적지초마디에 도착하며 목적지초마디는 호상접속으로부터 그 파케트를 제거한다.

다음 목적초마디우의 기억기부분체계는 자기의 국부기억기로부터 캐쉬행을 회복한다(되찾는다.). 동시에 목적지초마디는 하나의 접수통지파케트를 원천지초마디로 보낸다. 목적지초마디가 캐쉬행을 회복했을 때 그것은 자료를 포함하는 응답파케트를 원천지초마디로 보낸다.

마지막에 원천지초마디는 자료를 받았다는것을 가리키는 접수통지파케트를 목적지초마디로 보낸다.

Exemplar X체계의 기본특징은 자료전송을 가능하게 하는 특별히 설계된 크로스바와 자료이동기하드웨어를 사용하는것이다. 크로스바는 CPU들과 I/O장치들로부터 기억기부분체계로 비차단식접근을 제공한다.

처리기쪽의 8개의 크로스바포구 매개는 단일한 자료이동기에 접속한다. 자료이동기는 한쌍의 PA-8000처리기를 지원하며 3개의 PCI확장홈 및 CTI에로의 450MB/s의 대면부를 제공하는 240MB/s의 I/O통로를 지원한다. 기억기폭크로스바의 매 포구는 4통로끼워넣기식기억기기판에 접속한다. 크로스바포구의 자료경로폭은 64b이다. 그러므로 크로스바포구의 최대대역너비는 매 방향에서 960MB/s이며 전체 크로스바의 집체대역너비는 두개 방향을 계산하면 15GB/s이다.

디스크들과 테프들, RAID디스크, 망과 같은 여러가지 I/O장치들은 PCI확장홈을 통하여 임의의 초마디에 련결할수 있다.

8．3．2．Exemplar쏘프트웨어환경

Exemplar X-Class의 쏘프트웨어환경을 그림 8-8에서 보여 준다.

X-Class체계는 많은 순차프로그람들을 동시에 실행하며 큰 처리량을 제공하는 거대한 워크스테이션으로 볼수 있다. 이 특징은 점점 더 많은 현대적인 병렬체계들에 적용되고 있다.

SPP-UX조작체계

조작체계는 SPP-UX라고 부르는데 이것은 Unix의 HP판본인 HP-UX의 확장으로 볼수 있다.

HP-UX 응용들	Exemplar 병렬응용들
HP-UX Middleware 와 체계도구들	병렬콤파일러도구들과 대규모체계의 특징들
API	병렬프로그람작성모형
SPP-UX 확장가능 Unix 극소형핵심부	

그림 8-8. SPP-UX조작체계와 쏘프트웨어환경

주목할만한 특징은 Exemplar병렬가동환경이 순차응용들을 지원한다는것이다. 사실상 SPP−UX는 HP워크스테이션에서 보게 되는 HP−UX와 꼭 같은 환경(그림 8−8의 그늘진 부분에서 보여 준다.)을 제공한다. 그러므로 많은 순차응용들, HP워크스테이션과 SMP봉사기의 가동환경우에 존재하는 2진코드들은 그 어떤 수정이나 재콤파일이 없이 실행될수 있다.

SPP−UX는 매개 초마디의 극소형핵심부를 실행하며 극소형핵심부는 가상기억기관리와 처리기의 일정짜기와 같은 가장 근본적인 핵심부기능을 제공한다.

조작체계의 대부분은 봉사기과제들과 사용자공간에서 실행하는 보호모듈들에서 실현되며 파일체계관리와 장치관리, 망봉사들과 같은 표준기능들을 제공한다.

SPP−UX는 테라바이트파일들과 파일체계, 64b자료형과 포인터들을 지원하는 64b특징들, 묶음일감일정짜기, 검사점, 유연한 자원배정기능과 같은 대규모체계의 특징들을 제공한다. 이 특징들은 자료쎈터들이나 기업체계들에서 필요하다.

병렬프로그람작성모형

SPP−UX는 단일병렬프로그람작성모형을 제공하는데 이것은 C와 C++, Fortran 77, Fortran 90, 공유기억기 및 통보문넘기기프로그람작성을 위한 콤파일러지령들과 서고들을 지원한다. 지원하는 서고에는 PVM, MPI, Pthreads들이 포함된다.

병렬프로그람은 실행시에 사용자가 리용할수 있는 처리기의 수에 자동적으로 자체 적응할수 있다.

프로그람이 실행을 시작할 때 리용가능한 매개 처리기당 1개의 스레드가 있으며 스레드 0(**뿌리스레드** 또는 **기초스레드**라고 한다.)을 제외한 모든 스레드들은 휴식한다.

스레드 0은 병렬구문을 만날 때까지 혼자서 실행한다. 그다음 그것은 프로그람의 모든 휴식스레드들을 활성화한다. 병렬구문의 마지막에 스레드들은 장벽(barrier)에 동기화되며 스레드 0을 제외한 모든 스레드들은 휴식하게 되며 일감을 기다린다.

콤파일러지원

Exemplar콤파일러는 자동최량화의 계층을 제공한다.

제일 낮은 수준은 코드생성수준이다. 여기서는 지령일정짜기, 쏘프트웨어관흐름화, 타일식등록기배정, 고리차단(loop blocking) 등과 같은 발전된 RISC최량화기술들에 의해 스칼라성능이 높아 진다.

다음수준은 자료독립고리들의 자동병렬성이다.

다음수준은 병렬성과 자료분배를 명백하게 조종하기 위한 콤파일러지령들을 리용한다.

도형과 응용도구

SPP−UX환경은 부분복합체관리자(Subcomplex Manager)라고 하는 도형도구를 포함하는데 이것은 처리기들을 부분복합체(다른 체계들에서는 pool 또는 partition이라고 부른다.)라고 하는 여러개의 무리(group)로 동적으로 구획 짓도록 한다.

실례로 64개의 처리기들을 4개의 부분복합체들로 즉 하나는 호상작용하는 일감들에, 하나는 묶음일감들에, 하나는 파일봉사기들에, 하나는 시각화에로 구획 지을수 있다.

부분복합체내에서 통보문넘기기는 공유기억기를 통하여 실현된다. 부분복합체들사이의 통보문넘기기는 전통적인 Unix Sockets를 통하여 실현된다.

SPP-UX는 또한 Cxtools라고 하는 환경도구들의 조를 제공하는데 이것은 결함수정과 검사, 병렬응용의 최량화에서 사용자를 돕는다. 이 도구조는 병렬결함수정(parallel debugger) 즉 병렬성능 profiler/해석기와 병렬사건해석기를 포함한다.

8. 4. Sequent의 NUMA-Q 2000

Sequent는 1992년에 NUMA-Q설계를 시작하였으며 첫번째 NUMA-Q 2000체계는 1996년에 Fall에 적재되였다. 이 체계는 주로 상업응용들을 위하여 설계되였는데 CC-NUMA 기계구성방식을 받아 들였다. 그 독특성은 대규모체계제작을 위해 상품적구성요소들과 표준구성요소들을 효과적으로 사용한다는데 있다.

이것은 Sequent가 자원들을 기능과 성능에 크게 영향을 주지만 아직 상품은 없는 그러한 주요구성요소들을 개발하는데 사용할수 있게 한다. 한가지 실례는 IQ-Link라고 하는 SCI대면부이다.

8. 4. 1. NUMA-Q 2000의 구성방식

NUMA-Q 2000구조는 그림 8-9에서 설명되며 그 중요한 구조적파라메터들은 표 8-4에 기입된다. NUMA-Q의 기본제작블로크는 Quad라고 하는 4개 처리기 SMP기판이다. 체계는 63개까지의 Quad 또는 252개의 처리기들을 가질수 있다. 이것은 CC-NUMA기계이다.

Quad들에서 모든 국부기억기들은 임의의 처리기로 접근가능한 대역기억기를 형성하는 SCI식호상접속(IQ련결들과 하나의 IQ-Plμs)에 의하여 한데 붙어 있다. 캐쉬일관성은 하드웨어목록식규약에 의해 실시된다.

Intel SHV Quad의 사용

Sequent는 NUMA-Q 제작을 위해 상품적구성요소들과 표준구성요소들을 널리 사용하고 있다.

Quad는 off-the-shelf Intel SHV봉사기기판의 변종이다.

표 8-4 Sequent NUMA-Q 2000자원의 속성들

속성	Quad	체계
처리기의 최대수	4	252
캐쉬기억기용량(MB), L2은 처리기묶음우에 있는것, L3은 원격자료캐쉬	L2:0.512 L3:32	L2:3.15 L3:2016
물리기억기용량(GB)	0.5-4	31-252
집체최대기억기대역너비(GB/s)	0.533	33.5
I/O포구의 수	PCI 7개	441
집체최대 I/O대역너비(GB/s)	0.266	16.7

NUMA-Q의 Quad는 64b, 66MHz국부모선으로 접속한 4개의 Pentium Pro처리기와 한개의 국부기억기로 구성된다. 또한 국부모선에 접속한 2개의 Intel Orion PCI다리들(OPB)과 IQ-Link라고 하는 1개의 SCI대면부기판이 있다.

이 32b, 33MHz PCI모선들 매개는 4개의 확장홈들을 가지며 총 8개의 확장홈을 가진다. 한개의 확장홈은 PCI-EISA다리를 통하여 관리 및 진단처리기(MDP)에 접속되며 리용가능한 PCI확장홈으로는 모두 7개가 남는다.

Intel의 4개 처리기기판은 CC-NUMA체계에서의 사용을 쉽게 하는 여러개의 특징들을 제공한다.

실례로 그 기판은 공개된 명세서를 가지는 열린 체계이다. 기판은 3부류의 지정된 모선발동자가 국부모선을 조종하게 한다. NUMA-Q에서 발동자는 IQ-Link카드이며 여러개 Quad들을 큰 체계로 통합하는 교착제이다.

기판은 처리기들이 봉사하고 있는 단 하나의 기억기요청으로 차단되지 않도록 여러개의 특수한 기억기요청들을 지원하는 능력을 제한하였다.

성능을 개선하고 SCI와 통합하기 위하여 Sequent는 Intel Quad기판에 일련의 수정과 보강을 하였다. 여유 있는 전원공급과 더 큰 고장분리능력이 보충되였다. 사용되지 않는 PC론리는 기판에서 제거되였다.

체계호상접속

그림 8-9와 같이 5가지 형태의 표준적인 체계호상접속이 있다.

그중 4개는 상품적호상접속으로서 2개의 PCI모선을 통하여 Quad를 련결한다. IQ-Link호상접속은 Sequent의 소유물이지만 SCI표준을 따른다. 그것은 Quad의 국부모선에 직접 접속되며 분산공유기억기를 위한 Quad호상간의 통신을 지원한다.

ㄱ) 4개 처리기 Quad

ㄴ) 여러개의 Quad를 리용하는 NUMA-Q2000

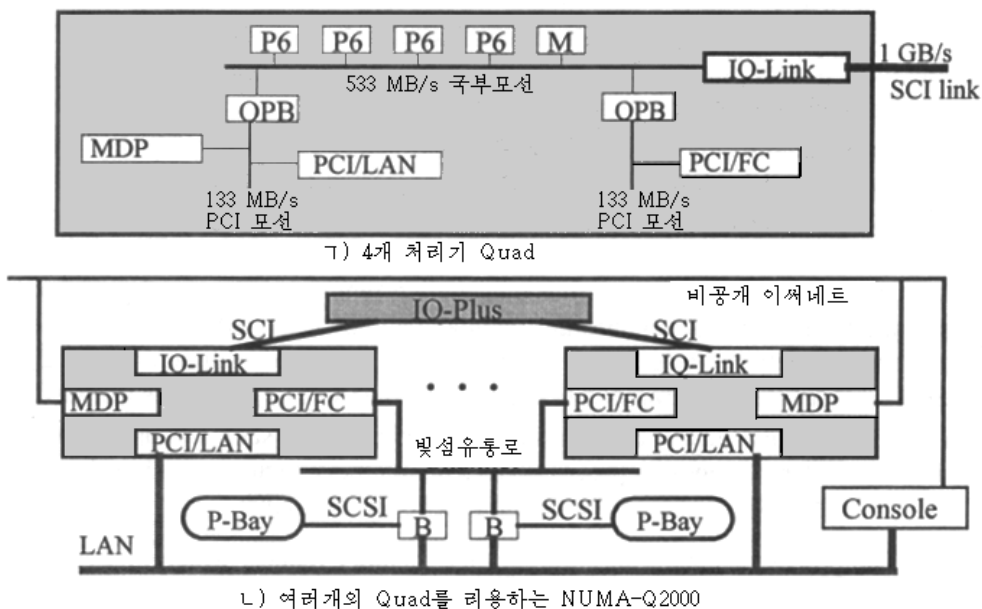



그림 8-9. NUMA-Q 2000의 구성방식도

임의의 IQ-Plμs는 중앙통(central Box)에 통합한 SCI고리이다. 대신에 개별적인 련결들은 IQ-Link들을 하나의 고리로 접속하는데 사용될수 있다. IQ-Link들과 IQ-Plμs를 접속하는 련결들은 15m까지의 동케블이다.

비공개이써네트는 체계 console을 모든 Quad의 Sequent가 설계한 MDP (management and diagnostic processor)에 접속한다. 이것은 Quad들에 대한 조종과 검사, 감시를 쉽게 한다.

공개된 LAN(이써네트, 고속이써네트, FDDI 등)은 Quad와 console을 바갈세계와 접속한다. 의뢰기 PC들과 워크스테이션말단들은 LAN을 통하여 NUMA-Q체계에 접근할수 있다.

대규모기억장치들(디스크들, RAID, 테프들, CD-ROM들)에로의 접속은 하나 또는 그 이상의 100MB/s빛섬유통로에 기초하며 매 빛섬유통로는 500m까지의 선택적인 케블이다.

Sequent가 개발한 빛섬유통로다리들은 빛섬유통로를 기억장치들이 련결되여 있는 빠르고 넓은 SCSI모선들에 접속한다.

IQ-Link

NUMA-Q에서 기본적인 기술혁신은 CC-NUMA구조를 효과적으로 실현하는데서 결정적인 SCI대면부기판(IQ-Link는 그림 8-10에서 보여 준다.)이다. 여기에서도 Sequent는 가능한껏 off-the-shelf구성요소들을 사용한다.

물리적으로 IQ-Link는 4개의 구성요소 즉 LSI Logic의 Orion모선대면부조종기(OBIC)소편과 SCI캐쉬련결대면부조종기(SCLIC)소편, Vitesse Semiconductor의 DataPump소편, 동기 DRAM렬들로 구성된다.

SCLIC소편만이 Sequent에 의해 전용으로 만들어 졌다.

DataPump는 500MHz로 동작하는 GaAs소편이다.

다른 구성요소들은 CMOS기술을 사용하며 66MHz의 국부모선박자속도로 동작한다.

3개의 소편은 사소한것이 아니며 매개가 140000개이상의 문들과 550개정도의 핀들을 포함한다.

론리적으로 IQ-Link는 3개의 부분체계들로 구성된다.

DataPump는 직접 SCI망을 돌리며 SCI표준에서 제기한 대면부와 류사하다(그림 6.37을 참고). 모선쪽대면부는 원격캐쉬의 자료배렬들을 관리하며 모선snooping론리를 관리한다. OBIC소편은 4개까지의 특수한 원격접근과 2개의 들어 오는 원격요청을 지원한다. 목록조종기는 SCI의 망쪽국부기억기목록과 망쪽원격꼬리표(tag)들을 관리하며 그것들을 목록에 기초한 캐쉬일관성규약에서 사용한다.

프로그람가능한 규약엔진(programmable protocol engine, SCLIC)이 이 과제를 수행한다.

원격기억기요청은 처리기나 I/O조종기들중 하나와 같은 임의의 모선발동자에 의해 나올수 있다.

처리기가 원격마디로부터 자료를 꺼내는데 필요한 기억기요청을 내보낸다고 하자. 이때 그것은 다음의 걸음들을 거친다.

(1) 처리기는 캐쉬 L1과 L2에서 실패하는 묶음호상기억기요청을 만든다.
(2) 처리기는 실패캐쉬행을 회복하기 위하여 기억기읽기요청을 기억기모선에 배치한다.

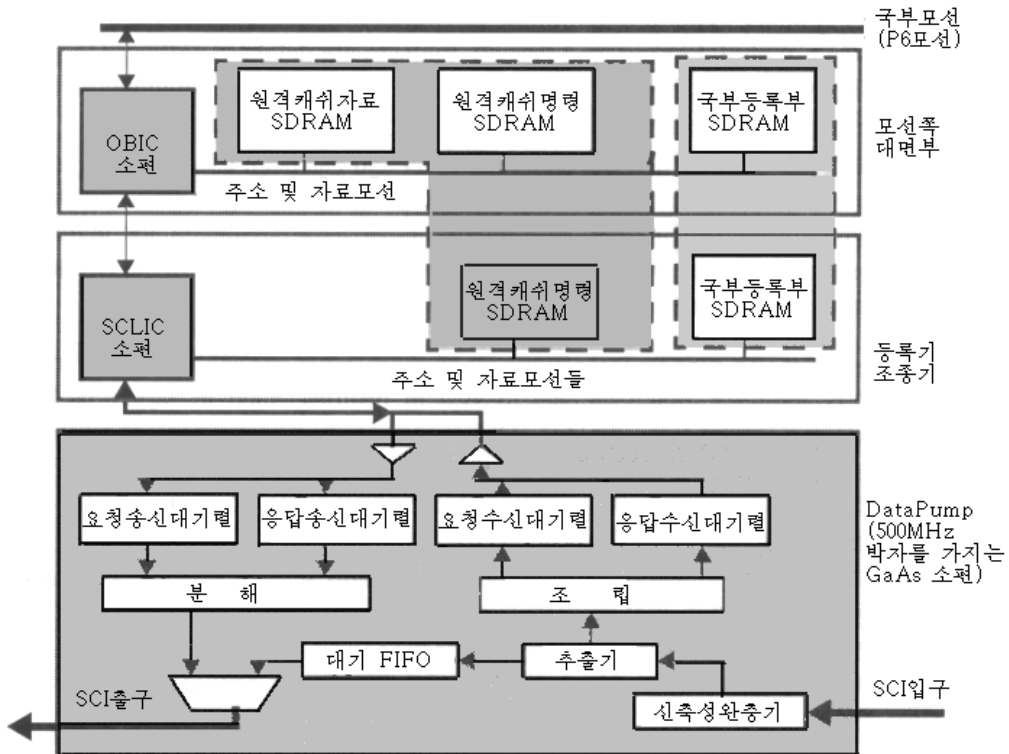



그림 8-10. IQ-Link의 대략적인 블로크도

(3) OBIC는 주소가 국부기억기범위내에 있는가를 검사하는 하나의 snoop주기를 실행한다. 주소가 국부기억기공간의 밖에 있으므로 원격캐쉬의 꼬리표들이 검사된다.

(4) 요청은 원격캐쉬에서 성공하지 못하며 기억기모선에서 OBIC참조가 즉시에 완성되지 않는다는것을 가리키게 한다. 이것은 다음의 독립적인 트랜잭션들이 기다림이 없이 진행되도록 한다. 다음 OBIC는 원격캐쉬의 하나의 선을 배정하고 요청을 SCLIC로 통과시킨다.

(5) 규약엔진은 CACHE-FRESH지령과 캐쉬행의 주소, 행의 home마디의 ID를 포함하며 하나의 파케트를 구성하는 하나의 과제를 생성한다(낳는다.).

(6) 과제는 파케트를 DataPump로 통과시킨다. 다음 과제는 내부의 꼬리표캐쉬에 하나의 선을 배정하고 잠자기에 들어 간다.

(7) DataPump는 요청파케트를 SCI고리를 거쳐서 home마디로 넘기기한다. 본래의 마디는 자료를 검색하여 응답을 돌려 보낸다.

(8) home마디로부터 자료를 포함하는 응답이 도착할 때 DataPump는 응답을 포착하고 그것을 목록조종기로 보낸다.

(9) 응답은 자체의 트랜잭션ID를 사용하여 걸음 (6)에서 잠자기에 들어 간 과제를 깨우고 그 자료를 OBIC로 보내며 캐쉬상태를 갱신하고 끝낸다.

(10) OBIC는 모선을 중계한 다음 응답을 기억기모선에 배치한다. 동시에 자료는 원격캐쉬로 써지며 캐쉬상태가 갱신된다.
(11) 걸음 (2)에서 시작한 기억기읽기트랜잭션은 끝나고 캐쉬행은 L1/L2캐쉬들과 적당한 등록기로 적재된다.

8．4．2．NUMA-Q의 쏘프트웨어환경

NUMA-Q기계는 DYNIX/ptx를 실행한다. DYNIX/ptx는 다중처리기체계들로 된 Symmetry와 NUMA-Q 2000계렬을 지원하기 위하여 개선된 Unix조작체계의 Sequent판본이다.

프로그람작성환경은 표준적인 순차언어들(C, C++, Cobol, Fortran, Pascal)과 대형콤퓨터자료기지쏘프트웨어(실레로 Informix, CA/Ingres, Oracle, Progress, Sybase, Unify), ptx/Debug라고 하는 병렬결함수정기, 병렬프로그람작성을 위한 포괄적인 서고루틴들의 모임을 포함한다.

DYNIX체계는 1984년에 처음으로 개발되여 Symmetry다중처리기계렬에서 10번이상 사용되였다. 그것은 수천의 사용자와 수TB의 자료를 가지는 대규모관계자료기지응용을 지원할수 있다.

DYNIX/ptx체계는 상업적인 기업응용을 위하여 설계되였으므로 확대가능성과 유용성, 다루기쉬움을 위한 여러 특징들을 제공한다.

확대가능성지원

수많은 사용자들과 대규모응용들, 대규모자료모임들을 지원하기 위하여 DYNIX/ptx는 더 큰 표들과 하쉬법과 같은 체계유틸리티들에 대한 보강을 포함한다. DYNIX/ptx는 16GB까지의 물리기억기와 수십테라바이트의 디스크기억을 지원하며 수천의 사용자들을 동시적으로 지원한다.

응답시간의 퇴보를 최소화하기 위하여 DYNIX/ptx는 작은 부하를 가지는 처리기들로 과제들을 자동적으로 분배하고 일정짜기하며 동적부하평형을 실현한다.

CC-NUMA구조와 SCI고리는 지연을 줄이고 대역너비를 늘이도록 한다.

순차일정짜기는 무리일정짜기(처리기구룹)의 형식과 프로쎄스들이 자기의 자료에 가깝도록 일정 짜는 노력들을 가능하게 한다.

마지막으로 단일한 NUMA-Q기계가 제공할수 있는것보다 더 높은 확대가능성이 요구되면 여러개의 체계들이 ptx/CLMSTER쏘프트웨어를 사용하여 하나로 클라스터화될수 있다.

유용성지원

순차볼륨관리자는 NUMA-Q 2000주변장치들에서 디스크들과 테프들의 직결교체를 가능하게 한다. 이것은 또한 디스크반사화능력과 여분의 boot디스크들, 뿌리와 교환장치들을 반사하는 능력을 제공한다.

개선된 파일체계는 파일체계회복시간을 줄인다. 파일체계는 5s내에 회복될수 있다.

더 높은 유용성은 대등한 오유회복을 제공하는 분산잠금관리자와 클라스터무결관리자를 포함하는 ptx/클라스터쏘프트웨어에 의해 제공된다.

다루기쉬움

Sequent는 주콤퓨터부류를 다루기 쉽게 하기 위한 여러개의 도구들을 제공한다. 이 도구들은 다음과 같은것들을 포함한다.

- 체계설치와 디스크복제, 쏘프트웨어설치, 사용자재정관리, 구성관리 등을 일상적으로 처리하는 ptx/CLMSTER라고 부르는 안내에 기초한 체계관리.
  ptx/CLMSTER대면부는 자료기지체계기동이나 체계완료와 같은 사용자정의함수들을 허용함으로써 확대가능하다.
- 고객싸이트에서 NUMA-Q기계를 Sequent봉사기쎈터에 접속하는 SequentLINK라고 부르는 원격체계감시.
  이것은 Sequent기사들이 체계의 일상적인 동작을 원격으로 감시하고 문제를 원격으로 찾으며 봉사 및 구성요구들을 원격으로 제기하도록 한다.
- SNMP규약에 기초한 망화된 체계를 관리하는 CommandPoint라고 부르는 통합된 쏘프트웨어모임.
- 여러개의 고성능테프구동기를 지원하며 완전복제, 증식복제, 복제시효, 파일목록들 등을 관리하는 Ptx/ESBM이라고 부르는 개선된 디스크복제쏘프트웨어.

Sequent NUMA-Q는 열린체계이며 일감관리, 자료기지관리, 망관리 등을 위한 쏘프트웨어와 같은 많은 3부류쏘프트웨어묶음들(package)을 지원한다.

8. 4. 3. NUMA-Q의 성능

Sequent는 Symmetry 2000과 5000에 대한 TPC-B와 TPC-D성능평가기준을 실행하고 성능자료들을 모으기 위한 하드웨어계수기, 론리해석기, 조작체계4진계수기를 사용하여 NUMA-Q 2000의 성능을 예측하였다.

다음 여러가지 모의결과들을 생성하는 모의모형이 이 성능자료들에 기초하여 만들어 졌다.

Sequent는 OLTP와 결심지원체계(DSS)응용들의 작업부하특성들이 전형적이라고 보고 있는데 모의결과는 NUMA-Q동작의 작업부하가 그러한 작업부하들보다 낮다는것을 예측하고 있다. 예측된 NUMA-Q체계는 32개의 133MHz Pentium Pro처리기들(8개의 Quad들)을 사용하며 여기서 매개 처리기는 512KB의 2차캐쉬를 가진다.

NUMA-Q개발에서 진행된 관찰결과는 상업응용들에서 자료접근의 대부분이 큰 (4GB의)국부기억과 큰 (32MB의)원격캐쉬를 가지는 하나의 Quad내에서 충족될수 있다는것이다. 결과 Quad내의 망(SCI고리)에서 대부분의 통신흐름은 캐쉬일관성규약(무효)을 위한것이

지 자료전송을 위한것이 아니다. 그리하여 체계가 제공하는 자료전송대역너비는 모든 국부Quad자료대역너비들의 합과 거의 같다.

이것이 일반적으로 성립되면 통신대역너비가 중요한 제한요인이 아니라 지연숨김이 더 중요하게 될것이다.

작업부하특성

캐쉬접근동작을 표 8-5에서 보여 준다.

32개 처리기 NUMA-Q는 64B의 캐쉬행들과 32MB의 원격캐쉬들, 512KB의 2차캐쉬들을 가진다.

TPC-B에 대하여 10000개 지령들을 수행하는데 평균 228개의 2차캐쉬실패가 생성된다. 그중 20개의 실패는 캐쉬행을 가지는 모든 Quad들에 보내지는 무효(invalidation)에 원인이 있다. TPC-D에 대하여서는 18개의 실패만이 있으며 1.6개의 캐쉬행이 무효된다.

TPC-B에서 L2실패의 50.4%는 국부 Quad기억기에서 자료를 찾고 27.2%는 같은 Quad안의 원격캐쉬에서, 1.3%는 같은 Quad의 다른 L2캐쉬에서 찾게 된다.

이 모든 경우에 무효로 해서 Quad호상간의 통신(8.9%)이 여전히 필요하지만 Quad호상간의 자료전송은 요구되지 않는다.

TPC-D렬은 TPC-D Quad-6에 대한 값들을 보여 준다.

이 2개의 성능평가기준은 완전히 서로 다른 특징을 가진다.

TPC-D는 자료창고, 결심지원체계의 성능평가기준인데 드문히 대규모의 정규자료묶음들에 대한 읽기접근들이 기준으로 된다.

표 8-5 32개 처리기 NUMA-Q에서 TPC-B와 TPC-D성능평가기준들에 대한 캐쉬접근분석표

캐쉬사건	속도	PTC-B	TPC-D
L2캐쉬실패	지령당	0.0223	0.0018
국부기억기성공	L2 실패당	50.4%	51.6%
원격캐쉬성공	L2 실패당	27.2%	27.6%
국부캐쉬 대 캐쉬전송	L2 실패당	1.3%	1.6%
2-hop원격	L2 실패당	3.1%	3.6%
4-hop원격	L2 실패당	9.1%	10.9%
국부성공, 원격무효	L2 실패당	8.9%	4.5%
공유목록길이	공유선에로 쓰기당	1.3	1.6
무효선	지령당	0.002	0.00016
원격무효	무효당	96%	109%

TPC-B성능평가기준은 [501]에서 서술된 TPC-C성능평가기준과 비슷하다. 그것은 작은 자료블로크들(자료기지의 레코드들)에로의 많은 read/write접근을 만든다.

결과적으로 TPC-B는 TPC-D보다 더 낮은 L2캐쉬실패률을 가지며 TPC-B는 더 많은 무

효동작들을 수행하여야 한다. 2개의 성능평가기준의 다른 특징들은 류사하다.

모의결과

모의결과는 자료전송대역너비가 문제로 되지 않는 NUMA-Q는 평형된다는것을 보여 준다. Quad에서 국부모선의 리용은 40%를 넘지 않으며 SCI고리의 리용률은 33%이하이다. L2캐쉬실패들에서의 평균지연(즉 L2캐쉬실패벌금)을 그림 8-11에서 보여 준다. 대부분의 경우에 지연은 2μs이하이며 클라스터들이나 MPP들에서의 지연들보다 더 높다.

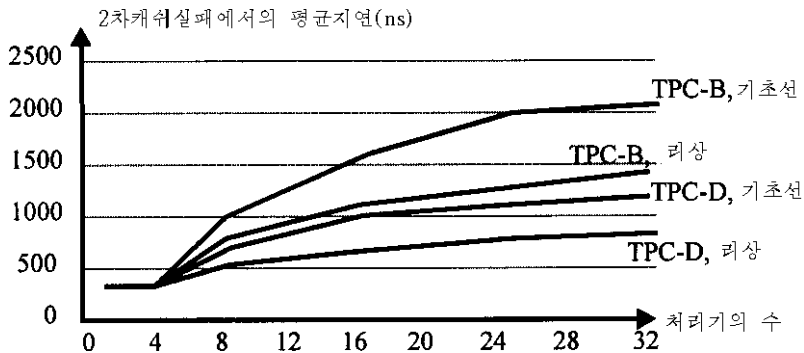



그림 8-11. NUMA-Q에서 실행하는 TPC성능평가기준의 지연

그림은 또한 전용설계된 하드웨어에서 캐쉬일관성규약의 처리는 여전히 큰 지연을 가져 온다는것을 보여 준다.

NUMA-Q의 속도증가형태는 그림 8-12에서 보여 준다.

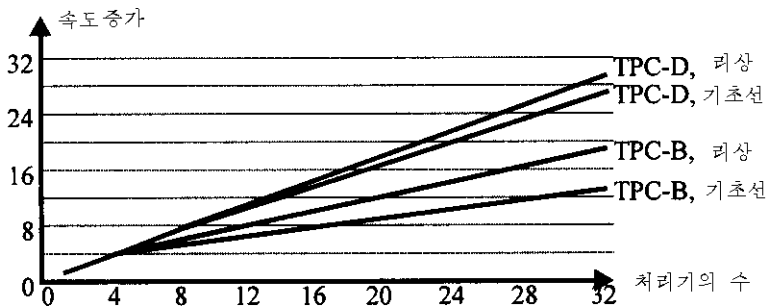



그림 8-12. 하나의 처리기에 대한 NUMA-Q성능

더 큰 지연으로 하여 TPC-B성능평가기준은 TPC-D성능평가기준보다 더 나쁜 성능을 보여 준다. TPC-B는 TPC-D성능평가기준보다 10배 더 높은 L2캐쉬실패률을 가지기때문에(지령당 0.0223대 0.00018, 표 8-5를 참고) 효률적인 캐쉬실패처리는 더욱 중요하다.

특히 그림 8-12는 캐쉬일관성규약처리를 개선하는것이 TPC-B의 성능을 43%만큼 더 올릴수 있다는것을 암시한다. Sequent는 SCLIC에서 2개의 규약엔진사용을 연구하고 있는데 그 모의는 TPC-B성능을 20% 개선할수 있다는것을 보여 주고 있다.

8. 5. SGI/Cray Origin 2000초고속봉사기

SGI/Cray는 1996년 10월에 Origin 2000체계를 발표하였다. 이 체계계렬은 CC-NUMA기계를 받아 들였다. SGI/Cray에서 이 구조는 내적으로 확대가능공유기억기다중처리(Scalable Shared-memory Multiprocessing, S^2MP 또는 S2MP)체계로 알려 져 있다.

먼저 설계목표들을 고찰한 다음 확대가능한 콤퓨터를 위하여 개발된 구조와 쏘프트웨어환경들을 론의한다.

8. 5. 1. Origin 2000계렬의 설계목표

Origin 2000체계는 다음의 특별한 판매목표들을 가지는 고도로 확대가능한 초고속봉사기로서 설계되였다.

- 체계는 1개 처리기로부터 1024개 처리기까지의 크기로 확장할수 있으며 이에 비례되는 자료기억과 기억기접근, 통신, 동기화, I/O능력들을 유지한다. 그러나 128개까지의 처리기로 된 체계들만이 1997년에 완성되였다.
- 체계는 현존응용들을 지원하기 위하여 Power Challenge봉사기계렬의 캐쉬일관성의 대역적으로 주소화가능한 기억기모형을 유지한다.
- 입력된 자료수준과 증가가격은 높은급SMP의것보다 더 낮아야 하며 워크스테이션들의 클라스터의것에 접근하여야 한다.

이러한 목표들을 달성하기 위하여 SGI의 Origin설계팀은 Stanford Dash프로젝트에서 쌓은 경험에 기초하여 다음의 결정들을 채택하였다.

- 체계는 캐쉬일관성을 실시하는 공유기억기와 하드웨어를 가지는 CC-NUMA기계를 받아 들인다. 캐쉬일관성규약은 Stanford Dash기계에서 사용한 등록부에 기초한 규약의 개선된 판본이다.
- 모든 처리기는 모든 I/O포구들은 물론 체계안의 모든 기억기에 직접 접근할수 있어야 한다. 더우기 임의의 I/O장치는 국부기억기를 제외한 체계안의 모든 기억기에 대하여 DMA할수 있다.
- 체계는 기계크기에서의 늘어 나는 증대를 가능하게 하는 모듈식물리구조를 사용하여야 한다.
- 내부마디의 통신과 동기화, I/O에 대한 효률적인 하드웨어지원을 제공하여야 한다. 특히 원격의 기억기나 I/O장치들에 대한 접근은 국부접근보다 2배만큼 낮은 지연을 가져야 한다. 기억기와 I/O동작들에 대한 마디내부의 대역너비는 높아야 하며 집체대역너비는 기계의 크기에 따라 선형으로 커져야 한다.
- 체계는 SGI Power Challenge계렬과 뒤쪽 호환성이 있어야 한다.
- 조작체계는 CC-NUMA구조를 효률적으로 지원하여야 하며 SMP기계에서 이전에

는 없는 국부성과 유용성지원을 제공하여야 한다.

결과 Origin 2000구조는 1024개까지의 처리기들과 1TB의 주기억을 지원한다. 1~64개의 처리기를 가지는 체계들을 SGI Origin 2000이라고 부르고 128개이상의 처리기를 가지는 체계들을 Cray Origin 2000이라고 부른다.

8. 5. 2. Origin 2000의 구성방식

Origin 2000의 구조를 그림 8-13에서 설명하고 현재 리용가능한 구조적형태들은 표 8-6에서 개괄한다.

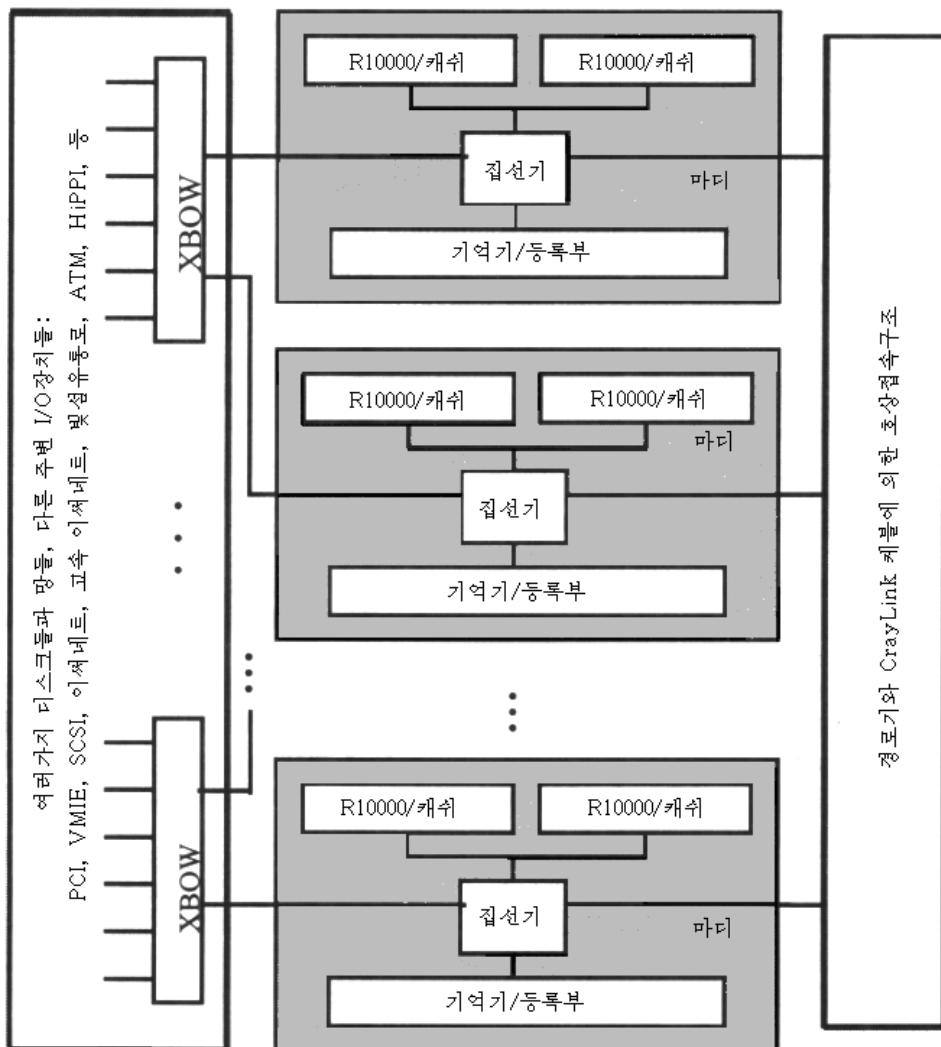



그림 8-13. Origin 2000다중처리기구조도

Origin 2000은 모듈식물리구조를 가진다.

최소deskside체계는 1~4개의 마디(1~8개의 처리기)와 1~32MB의 2차캐쉬, 64MB~16GB의 주기억기, 14개까지의 I/O조종기들을 포함한다. 마디들과 주기억, I/O조종기들, 호상접속은 체계에 점차 보충될수 있으며 결국에는 128개의 처리기들과 512MB의 2차캐쉬, 256GB의 주기억기, 208개의 I/O조종기들을 가지는 최대시렁(rack)체계로 확장될수 있다.

마디기판

매 마디는 4MB까지의 2차캐쉬를 가지는 박자가 195MHz인 1~2개의 MIPS R10000처리기들로 구성된다. 매개 R10000처리기는 한주기에 2개의 부동소수점연산(flop)을 할수 있으며 390Mflop/s의 최대속도를 가진다.

Origin 2000의 주기억은 물리적으로 마디들속에, 하드웨어는 캐쉬일관성을 등록부에 기초한 방법으로 실시한다.

Origin 2000은 그림 8-14에서 보여 주는것과 같이 갈라 져 있지만 모든 마디들에서 접근가능한 DSM이다. 기억기, 호상접속빛섬유, I/O부분체계를 접속하는데 집선기(HUB)라고 부르는 크로스바ASIC를 사용한다. 집선기ASIC는 4개의 쌍방향포구를 가진다. 195MHz박자에서 하나의 포구는 780MB/s의 한방향최대대역너비와 1.56GB/s의 완전2중최대대역너비를 제공한다.

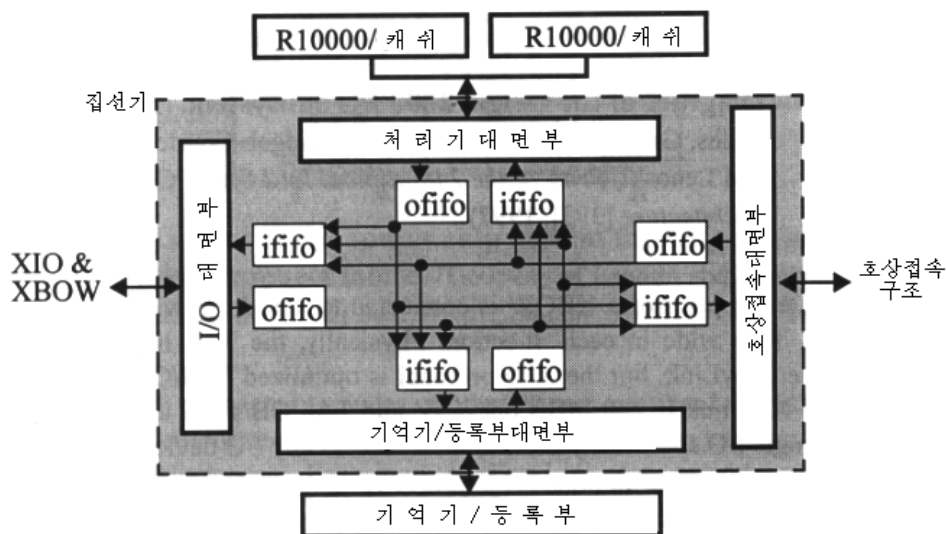



그림 8-14. Origin 2000마디의 크로스바집선기소편

집선기(HUB)는 마디내부 및 마디호상통신의 경로를 정하는데 사용된다. 이것은 내부의 통보문형식과 XIO 또는 CrayLink호상접속에 의해 사용된 외부형식사이의 호상변환을 담당한다.

호상변환은 대면부라고 부르는 4개의 개별적인 조종회로에 의해 진행된다. 매개 대면부는 2개의 FIFO를 가지는데 하나는 들어 오는 통보문들을 위한것(if:fo)이고 하나는 나가는 통보문들을 위한것(of:fo)이다.

Origin은 대역적인 실시간박자를 제공한다.

한개의 집선기는 박자소유자로 지정되는데 이것은 호상접속을 통하여 대역박자를 다른 집선기들에로 내보낸다.

I/O부분체계

Origin 2000 I/O부분체계는 208개까지의 I/O포구로 확장할수 있고 임의의 처리기로부터 접근가능하게 설계된다. I/O자료는 여러가지 장치들로 보내기 위하여 통보문으로 편성된다.

통보문형식과 경로정하기, 조종은 XIO라고 부르는 I/O규약에 의해 좌우된다. 매 마디기판은 하나의 XIO포구를 가지는데 XBOW라고 부르는 크로스바ASCI를 통하여 6개 포구까지 확대할수 있다.

표 8-6 SGI Origin 2000의 구조형태

속성	desksides	시렁(rack)
처리기의 수	1-8	2-128
최대속도(Gflop/s)	3.12	49.92
캐쉬기억기용량(MB)	1-32	2-512
물리기억기용량(GB)	0.064-16	0.064-256
집체최대기억기대역너비(GB/s)	3.12	49.92
I/O포구의 수	14	208
집체최대 I/O대역너비(GB/s)	6.2	102

I/O부분체계는 그림 8-15에서 설명된다. XBOW는 8개 포구크로스바교환기 ASCI이다. 모든 포구는 XIO규약을 사용한다.

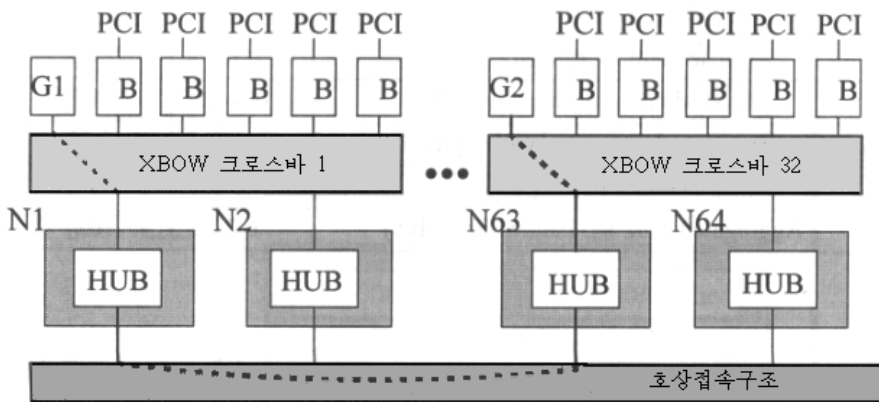



그림 8-15. Origin 2000 I/O부분체계의 도식

2개의 포구는 마디들을 접속하는데 사용되고 나머지 6개는 XIO장치들이나 widget라고 부르는 ASCI들에 직접 접속하는데 사용된다.

Widget들은 XIO규약을 도형장치나 PCI모선에서 사용되는 규약으로 전환하는 대면부이다. 32b 또는 64b PCI모선들을 통하여 ATM, HiPPI, 이써네트, FDDI, 빛섬유통로, SCSI, VME모선확장 등과 같은 서로 다른 형태의 I/O장치들이 Origin 2000에 접속될수 있다.

XBOW는 조종정보와 목적지정보를 결정한다.

XBOW의 8개 포구매개는 쌍방향 XIO련결에 의해 하나의 마디 또는 widget접속된다. 물리적으로 XIO련결은 Cray Link와 같은 기술을 사용하지만 XIO규약은 I/O통신흐름에 대하여 최량화된다. XIO련결의 최대대역너비는 한방향에서 780MB/s, 완전2중에서 1.56GB/s이다.

Origin 2000은 단일I/O공간을 제공한다. 즉 임의의 처리기는 임의의 I/O장치에 접근할수 있다.

실례로 마디 N1의 처리기는 자기의 국부집선기와 첫번째 XBOW를 통하여 도형장치 G1에 접근할수 있다. 만일 같은 처리기가 국부XBOW에 직접 련결되여 있지 않는 원격장치(실례로 G2)에 접근하려 한다면 그림 8-15에서 보여 주는바와 같이 국부집선기와 호상접속빛섬유, 마디 N63의 원격집선기, XBOW32를 거쳐서 그 장치와 통신하여야 한다.

호상접속구조

호상접속구조는 여러개의 마디들과 경로조종기들(그림 8-16)을 하이퍼립방체와 같은 위상(그림 8-17)에 따라 접속하는 CrayLink케블들로 이루어 진다.

매 CrayLink케블은 50개의 서로 다른 단일쌍들, 다 해서 100개의 전도체(선)를 포함하는 2중차페꼬임쌍축케블이다. 최대케블길이는 5m, 최소굴곡반경은 1.25인치이다.

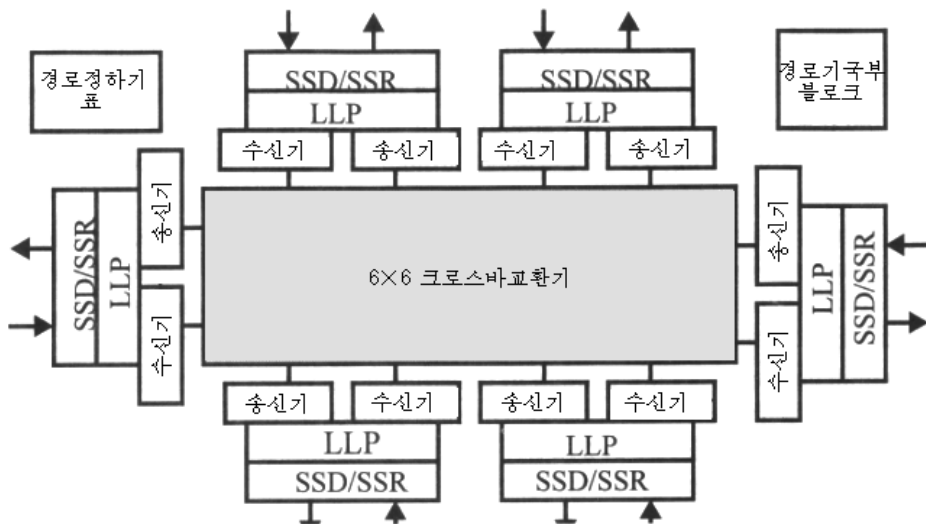



그림 8-16. Origin 2000호상접속구조에 사용된 SPIDER경로조종기소편

SGI SPIDER경로조종기소편은 통보문들에 대한 믿음직한 완충식wormhole경로정하기

를 제공한다. 이것은 state_of_the_art기술로 만든 6개 포구크로스바교환기이다. SPIDER소편은 백만개이상의 문들과 600개이상의 핀들을 포함한다.

매 포구는 박자가 390MHz인 한쌍의 한방향16b련결로 구성된다. 그러한 높은 주파수는 선진기술 즉 400MHz(2.5ns주기시간)만큼 높은 속도를 가능하게 하는 낮은 진동CMOS신호화기술인 차동PECL(positive−shifted ECL)과 STL(SGI transistor logic)에 의해 만들어 질수 있다.

이 포구들은 CrayLink케블들을 거쳐 마디들이나 다른 경로조종기들에 접속된다. 경로조종기는 6개의 포구모두가 완전중복으로 동시에 동작하도록 한다.

최종결과는 높은 통신성능이다. 매개 포구나 CrayLink는 780MB/s의 한방향최대대역너비와 1.56GB/s의 완전중복최대대역너비를 제공한다.

경로조종기소편내에서 6개 포구는 동등하다. 외부주파수가 390MHz라고 해도 경로조종기소편의 내부주파수는 단지 97.5MHz이다.

원천동기구동프로그람(source synchronous driver, SSD)과 원천동기수신기(source synchronous reciever, SSR)는 외부의 390MHz 16b자료와 내부의 97.5MHz 64b자료사이의 호상전환을 담당한다.

련결수준규약(LLP)회로는 믿음직한 통보문넘기기를 보장한다. CRC코드를 사용하여 오유가 발견되면 미끄러지는 창문규약을 통하여 재전송함으로써 정정된다. LLP에로의 자료전송을 제외하고 송신자는 하나의 신용도식(credit scheme)을 통하여 흐름조종을 관리한다. 수신자는 LLP로부터 받은 자료를 경로정하기표와 송신자에게 보낸다.

SPIDER경로조종기는 작은 하드웨어지연을 가지는 wormhole경로정하기를 실현한다. 핀대 핀 지연은 단지 41ns이다. SPIDER경로조종기는 또한 경쟁과 혼잡을 완화시키기 위한 여러가지 기술들을 사용한다. 매개 물리통로당 4개의 가상통로가 사용된다.

혼잡조종론리는 통보문들이 256개 수준의 통보문우선권이 지원되는 2개의 가상통로사이에서 융통성 있게 교환되도록 한다. 파케트의 우선권은 나이(차단된 시간)에 따라 높아져 우선권이 높은 파케트가 더 좋은 전송기회를 가지도록 한다.

굵은 하이퍼립방체위상

Origin 2000경로조종기의 또 하나의 흥미 있는 특징은 재설정(프로그람화)가능하다는것이다. 기정의 위상은 그림 8−17에서와 같이 하이퍼립방체와 같은 형태(굵은 하이퍼립방체라고 한다.)를 따른다. 그러나 호상접속구조안에서 경로조종기들의 경로정하기표들을 수정하여 다른 위상도 구성할수 있다. 특히 처리기들의 수는 그것의 제곱이 아니여도 된다. 경로조종기설정은 경로조종기안의 상태 및 조종등록기들을 통하여 진행된다.

굵은 하이퍼립방체는 전통적인 2진하이퍼립방체로부터 수정된다. 이 위상의 흥미 있는 속성은 그것이 하이퍼립방체의 선형2등분대역너비의 우월성을 유지하는 한편 마디차수의 증가를 피한다는것이다.

마디차수가 4일 때 16개 정점하이퍼립방체 또는 하이퍼립방체위상이 사용된다면 32개 마디 Origin 2000을 구성할수 있을뿐이다. 그러나 굵은 하이퍼립방체위상은 Origin 2000을 512개 마디까지 확장하게 한다. 이것을 어떻게 실현하는가의 일부는 아래에서 설명하고 일부는 독자들에게 련습으로 남긴다.

이것은 그림 8-17 ㄱ)장치들에 련결한 XBOW크로스바교환기에 접속된다.

SPIDER경로조와 같이 가장 작은 형태는 1개의 마디를 가지는데 마디는 2개의 처리기를 가지며 I/O종기(R)는 2개의 마디를 접속하며(그림 8-17 ㄴ) 두개의 경로조종기들은 4개의 마디를 접속한다(그림 8-17 ㄷ). 이 형태는 경로조종기들을 하이퍼립방체의 정점들로 볼 때 2진하이퍼립방체의 성장과 류사하게 32개 마디까지 보존된다(그림 8-17 ㅂ). 하이퍼립방체와의 차이는 다음과 같다.

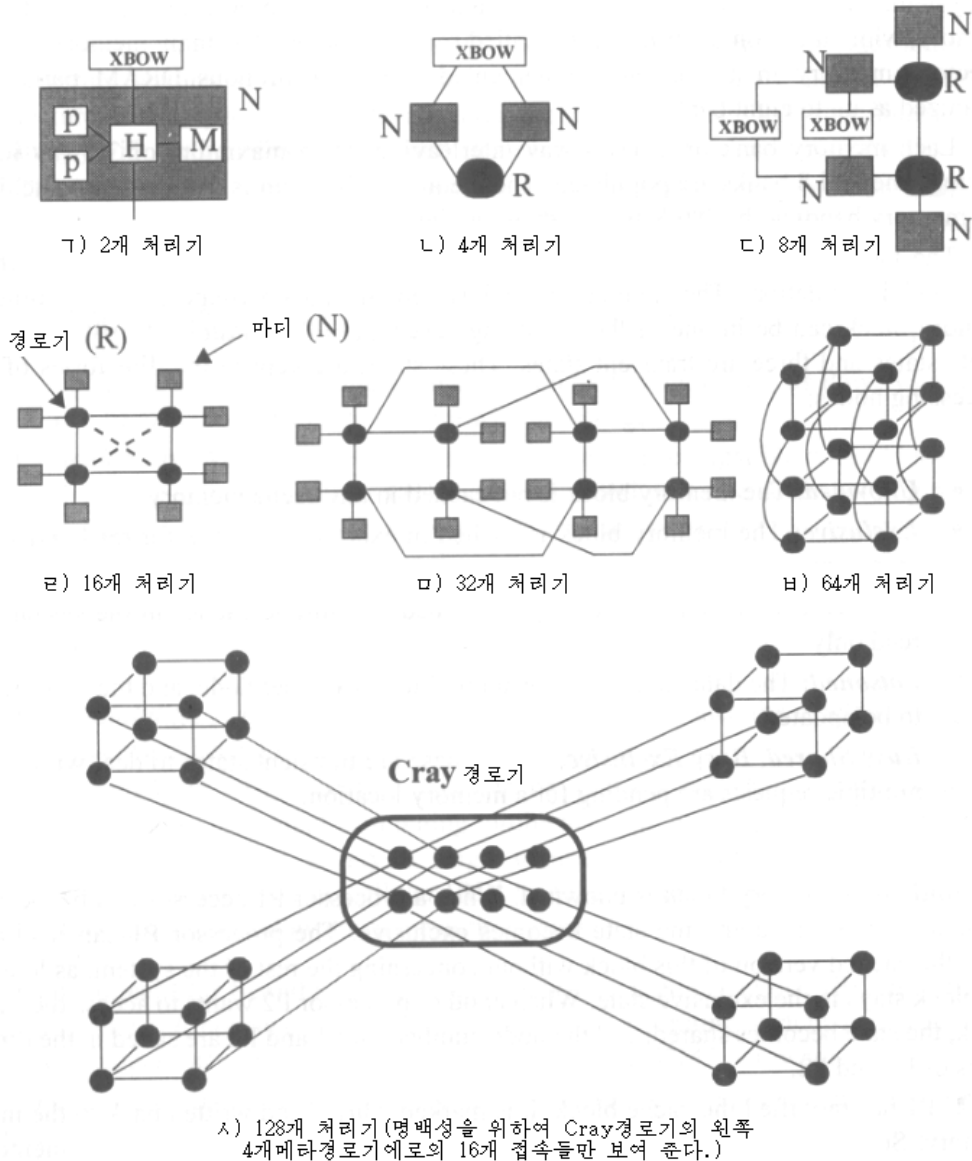



그림 8-17. Origin 2000에서 받아 들인 굵은 하이퍼립방체위상

(N:마디, M:기억기, P:처리기, H:집선기, R:경로조종기)

- Origin 2000의 위상은 《곤두 서 있으며》 여기서 2개의 마디는 하나의 경로조종

기에 접속된다.

- 사용되지 않는 경로조종기포구들은 지연을 줄이고 2등분대역너비를 늘이도록 확장련결들에 의해 접속될수 있다. 이것은 그림 8-17 ㄹ)에서 점선으로 설명된다.

32개 마디이상에서 Origin 2000은 나머지**경로조종기들(메타경로조종기**라고 부른다.)을 사용하는 굵은 하이퍼립방체위상을 받아 들인다. Cray경로조종기라고 부르는 하나의 통안에서 8개의 메타경로조종기가 물리적으로 실현된다. 이 메타경로조종기들은 임의의 마디들에 접속되지 않은 중간의 경로정하기를 위하여 단독으로 사용된다.

64개 마디(32개 정점 또는 경로조종기들) 굵은 하이퍼립방체위상은 그림 8−17 ㅅ)에서 보여 주는데 여기서 4개의 8개 정점부분립방체의 매 정점은 메타경로조종기에 접속된다. 4개 부분립방체들의 같은 모서리의 4개 정점들은 같은 메타경로조종기에 접속된다. 이 굵은 하이퍼립방체는 그림 8−17 ㅅ)의 매개 메타경로조종기들을 5차원하이퍼립방체로 교체함으로써 1024개까지의 처리기로 확장된다.

분산공유기억기

Origin 2000체계는 하나의 분산공유기억기부분체계를 가진다.

처리기로서의 같은 마디안의 기억기모듈들은 가까운 기억기, 다른 마디우의 기억기모듈들은 **원격기억기**라고 부른다.

한 마디안의 주기억기와 등록부기억기는 동기DRAM부분들을 사용하여 실현되며 8개까지의 층으로 조직된다. 매 기억기층은 4통로끼우기를 제공하며 8개의 층모두가 상주될 때 최대로 32통로끼우기를 제공한다. 기억기부분체계는 4통로끼우기를 가지는 충분한 기억기대역너비(780MB/s)를 달성하도록 충분히 빠르다.

하드웨어는 후쓰기와 무효에 의해 유지되는 등록부에 기초한 일관성규약을 실시한다. 기억기모형은 순차일치성이다.

임의의 시각에 기억기블로크는 다음의 7개의 캐쉬일관성상태들중의 하나에 있을수 있다. 그중 4개는 안정된 상태이며 3개는 불안정한 상태이다. 이 상태들은 프로쎄스마디들의 등록부들에 보존된다.

- **Unowned** 기억기블로크가 임의의 캐쉬기억기에서 꺼내 지지 않는다.
- **Exclusive** 기억기블로크가 하나의 처리기에 의한 읽기와 쓰기를 위하여 하나의 캐쉬안에서 정확히 꺼내 진다.
- **Shared** 기억기블로크가 읽기만을 위하여 체계안의 여러 캐쉬들안에서 꺼내 지게 된다.
- **Poisoned** 자료페지가 다른 마디로 이동되였으며 TLB가 갱신되여야 한다.
- **Busy Shared, Busy Exclusive, Wait** 이것들은 여러 요청들이 하나의 기억기위치에 의존하고 있을 때를 취급하는 불안정한 상태이다.

처음에 기억기블로크는 unowned이다.

처리기 P1가 이 블로크에 접근할 때 그것은 P1의 캐쉬로 옮겨 지며 상태는 exclμsive

로 된다. 처리기 P1는 그 블로크가 배타적인 상태에 있는 한 체계의 나머지에 상관없이 이 블로크의 캐쉬판본을 읽고 쓸수 있다.

다른 처리기 P2가 같은 블로크에 접근하려고 할 때 그 상태는 shared로 되며 P1와 P2의 마디번호들은 P1와 P2의 등록부들에 보존된다.

P1가 그 캐쉬블로크를 수정하면 그것은 dirty로 표식되며 주기억기로 후쓰기한다(written back). 그래서 P2는 낡은 복사대신에 주기억기로부터 가장 최근에 갱신된 자료의 복사를 적재한다. 2개의 처리기는 공유된 자료블로크를 각각의 캐쉬로부터 여러번 읽을수 있다.

그러나 P2이 블로크에 쓰려고 할때 그것은 무효통보문을 이 블로크를 공유하는 다른 모든 마디들에 보내야 한다. 이 정보는 P2의 등록부에서 유지된다.

일단 무효가 되면 P2캐쉬의 블로크는 다시 exclμsive로 되며 P2은 전진하여 거기에 쓸수 있다.

8. 5. 3. Cellular IRIX환경

Origin 2000은 새로운 기능들을 제공하며 SIMP구조를 효과적으로 개발하기 위한 여러가지 흥미있는 특징들을 제공하는 Cellular IRIX조작체계를 실행한다. 이것은 MPP들과 클라스터들의 우점(실례로 확대가능성과 유용성)을 가지는 SMP의 우월성(실례로 단일체계영상, 관리하기 쉬운)을 확장하고 있다.

1996년판본인 Cellular IRIX6.4는 모든 Origin 및 Onys2 Impact체계들에서 실행하는 64b의 다중스레드, 다중프로그람조작체계이다. Cellular IRIX의 설계목표는 다음과 같다.

호환성 Cellular IRIX는 X/Open, POSIX와 같은 열린체계표준으로 콤파일되며 SGI의 이전 IRIX5 및 IRIX6조작체계들과 호환된다. Power Challenge와 같은 SGI체계들우의 현존 응용들은 변경없이 Origin 2000우에서 실행될수 있다.

- **확대가능성** 동작환경은 이전의 체계들과 호환되지만 Cellular IRIX의 구조는 Origin 2000체계를 128 또는 그이상의 처리기들로 확장할수 있도록 재설계되였다. 대비적으로 SMP체계들은 보통 10개이상의 처리기에로 효률적으로 확장할수 없다. Cellular IRIX는 1TB의 가상기억기주소공간을 제공하며 9백만 TB만한 큰 파일들을 지원하는 64b조작체계이다. 이것은 Origin 2000이 대규모문제를 풀수 있게 한다.
- **유용성** 항상 SMP체계들의 약점으로 되여 온 유용성을 높이는데 여러가지 기술들이 사용된다.
- **처리량** Origin 2000은 대규모적인 개별적계산문제들을 푸는 초고속콤퓨터뿐만아니라 수많은 동시사용자요청들을 충족시키는 고성능망봉사기로 사용되는것을 목적으로 한다. 체계의 처리량을 높이기 위한 기술들이 Cellular IRIX에서 사용된다. 그래서 하나의 느린 일감이 전체 체계를 느리게 하지 않는다.

확대가능성을 위한 세포의 개념

Cellular IRIX에서는 확대가능성을 높이기 위한 2가지 방법 즉 조작체계핵심부를 모듈

화할수 있는 세포개념과 S2MP하드웨어를 효과적으로 개발하는 기억기관리쏘프트웨어가 사용된다.

Cellular IRIX구조는 핵심부본문(text)과 자료를 세포라고 하는 SMP크기의 덩어리들로 분할한다. 여기서 매개 세포는 핵심부본문과 자료구조에 대한 하나의 국부복사를 포함하며 처리기들과 기억기, I/O장치들의 모임으로 구성되는 기계모듈에 대한 조종을 담당한다.

세포개념의 주요한 특징은 조작체계의 봉사들이 분산되고 국부화된다는것이다. Origin 2000에서 실행하는 응용들은 모두 봉사를 위하여 중앙의 유일적인 핵심부로 가지 않는데 이것은 늘어 난 기계크기로 하여 병목으로 된다. 대신에 세포는 자기가 조종하는 기계모듈에 접근하기 위한 모든 봉사들을 제공한다. 자원들에 대한 잠금경쟁들은 보통 하나의 세포로 국부화된다.

여러개의 세포들이 하나의 Origin체계에 상주할수 있으며 이것은 응용들이 여러 기계모듈들에 동시적으로 그리고 독립적으로 접근할수 있게 한다.

세포호상간의 조정이 필요할 때 Cellular IRIX구조는 추가적인 자료복사나 문맥절환부가처리가 초래되지 않도록 담보한다.

기억기관리

Cellular IRIX는 S2MP하드웨어구조의 우월성을 가지는 새로운 동적기억기복제와 이동, 배치기구들을 포함한다. 기억기관리의 기본단위는 페지이다. Cellular IRIX는 사용자의 처리에 의해 동적으로 선택가능한 여러개의 페지크기(16KB, 64KB, 256KB, 1MB, 4MB, 16MB)들을 지원한다. 큰 페지크기를 사용함으로써 특히 대규모자료기지관리응용들에서 TLB(table lookaside buffer)실부가처리를 줄일수 있다.

기억기복제

이것은 페지를 공유하는 매개 처리기들에 가장 가까운 기억기위치에서 읽기용페지의 복사를 복제하는 기구이다. 페지가 읽기용일 때 하드웨어는 공유를 찾고 기억기복제를 수행하는 기억기관리부분체계를 호출한다.

기억기이동

이것은 처리기가 가장 빈번히 접근하는(읽기 또는 쓰기) 처리기에 가까운 페지를 옮기는 기구이다. 기억기페지가 접근되고 있을 때 하드웨어계수기들은 마디를 벗어 나는 참조들을 기록한다. 참조들의 수 또는 비률이 어떤 미리 정의된 상대적 또는 절대적인 턱값을 지나면 기억기관리부분체계는 그 페지를 이동한다.

이동은 큰 부가처리를 가지므로 Cellular IRIX는 지나친 이동을 막는 기술을 통합한다. 여기에는 반충검사(bounce detection)가 있는데 이것은 페지가 앞뒤로 이동되는것(탁구효과)을 막는다. 즉 어떤 페지들을 얼구고(즉 이동불가능) 필요한 때 그것들을 녹인다.

기억기배치

이것은 국부성을 실현하기 위한 자료와 스레드들의 초기배정이다. 이것은 체계가 제

공하는 기정값들이나 콤파일러지령들, 응용구동기억기방책(memory policies)을 사용하여 진행할수 있다.

Cellular IRIX환경은 사용자가 페지들을 다른 페지들과의 특정한 거리내에 배치하며 스레드들을 페지들로 배치하고 스레드들을 도형이나 I/O와 같은 하드웨어자원들로부터 특정한 거리에 위치시키도록 체계에 알릴수 있다.

기억기국부령역(Memory Locality Domains, MLD)

MLD는 가까이에 배치되는 가상페지들의 모임이다. 얼마나 가까운가는 MLD의 반경으로 측정되는데 이것은 1 + 교차한 경로조종기들의 수이다.

MLD사용의 개념을 실례 8.4에서 설명한다.

여러 MLD들을 하나의 MLDset로 묶을수 있는데 이것은 특별한 배치지령들에 의하여 기계위상에로 넘겨 진다. Cellular IRIX에서 방책모듈은 사용자가 방책(페지화, 배치, 복제, 이동 등)들이 하나의 주소범위에서 사용되여야 한다는것을 조작체계에 알릴수 있게 한다.

실례 8.4. MLD와 MLDset의 Origin 2000에로의 도입

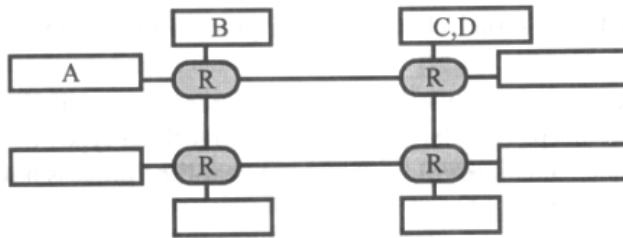



그림 8-18. MLD와 MLDset의 Origin 2000에로의 도입

프로그람은 8개 마디 Origin 2000에서 실행한다. 여기서 매 마디는 64MB의 기억기를 가진다. MLDset넘기기는 그림 8-18에서 보여 준다. 프로그람은 각각 48, 32, 24, 32MB의 4개의 큰 자료구조 A, B, C, D를 사용하는 6개의 스레드들을 가진다.

A와 B에 대한 접근들중 95%는 첫 4개 스레드들로부터, C와 D에 대한 접근들중 90%는 다른 2개의 스레드들로부터 생긴다.

이 자료로부터 2개의 MLD들로 구성된 하나의 MLDset를 정의할수 있다. 첫번째는 A와 B로 이루어 진 반경이 2인 MLD이고 두번째는 C와 D로 이루어 진 반경이 1인 MLD이다.

유용성지원

Origin 2000하드웨어는 ECC기억기와 여유전원 및 랭각, 즉시끼우기가능디스크들, RAID 등과 같은 유용성지원특징들을 가지도록 설계된다.

호상접속구조는 고장난 모듈을 분리할수 있다. 그리하여 일정한 체계부분은 나머지체계에 영향을 줌이 없이 정비를 위하여 전원이 차단될수 있다. 호상접속망은 마디들사이의

여러 경로들을 지원하며 결함있는 파케트들을 찾고 재전송하기 위한 하드웨어련결수준의 규약을 제공한다.

IRIX체계쏘프트웨어는 고속재초기적재를 위한 파일체계실행기록과 같은 유용성을 더욱더 높이기 위한 특징들을 포함한다. 즉 정지시간사건들과 지속, 증상들에 대한 감시와 통신을 위한 쏘프트웨어 availmon, 디스크반사화(mirroring)와 RAID, 응용실패방지를 위한 쏘프트웨어 IRIS FailSafe를 포함한다.

Origin 2000은 권한 없는 마디들이 기억기와 I/O장치들에 접근할수 없도록 접근보호규칙들을 제공한다. 세포개념은 조작체계봉사의 유용성을 지원한다. 이것은 우연적인 하드웨어 및 쏘프트웨어의 실패들을 하나의 세포내에 포함한다.

Cellular IRIX는 핵심부에 기초한 응용투명의 검사점재시동기구를 지원하는 얼마 안되는 조작체계들중의 하나이다.

사용자는 대화들과 프로쎄스들, 프로쎄스묶음들, 열린파일들, 관흐름들을 지원하면서 수정이 없이 현존 2진코드들을 검사할수 있다. 대기렬들과 sockets, 생디스크들, X−Windows, 도형들, 특수한 장치들의 지원을 통하여 고도로 리용가능한 망과 자료기지, 다른 응용들을 위한 보충적인 지원들이 계획되고 있다.

처리량지원−최량화된 일정짜기

Cellular IRIX는 체계들을 효과적으로 실현하고 처리량을 효과적으로 높이는 여러 기구들을 제공한다. 여기에는 수많은 처리기들과 사용자의 처리들을 지원하는 일정짜기와 높은 처리량과 담보된 I/O속도를 지원하는 64b실행기록파일체계 XFS가 포함된다.

핵심부는 일정짜기결정을 만들기 위하여 수미리초마다 중지된다. 프로쎄스들은 선점이 주어 진 매개 담보시간구간에서 실행된다. 시간구간의 마지막에 핵심부는 프로쎄스우선권에 기초하여 다음에 어느 프로쎄스를 실행하는가를 선택한다.

실행중의 프로쎄스는 또한 자기를 잠자게 하는 체계호출을 진행하여 자발적으로 다른 프로쎄스에 양보할수 있다.

Cellular IRIX6.4는 3가지 형태의 일감 즉 시간공유(호상작용), 실시간, 배경(묶음)을 지원한다. 실시간처리는 0(낮음)부터 255(높음)가지의 범위에 있는 우선권을 가진다. 배경처리는 항상 우선권이 0이며 자유CPU주기가 리용가능할 때만 실행된다. 호상작용의 프로쎄스는 0~40의 우선권을 가진다.

핵심부는 화페식기구에 기초하여 호상작용처리들의 일정을 짠다. 선택된 프로쎄스는 퇴화되지 않는 기정의 우선권 20을 가지고 그것이 체계호출이나 시간초과에 의해 잠자기에 들어 갈 때까지 처리기에서 실행된다.

화페식일정짜기기구는 매 사용자에게 우선권을 배정하는 전통적인 Unix일정짜기의 결함으로 하여 유도되였다. 새롭게 생긴 프로쎄스들은 같은 우선권을 이어 받으며 이것은 더 낮은 우선권의 프로쎄스들이 실행할 기회를 가지도록 프로쎄스가 실행됨에 따라 낮아 진다. 이 방법으로 상대적으로 높은 우선권을 가지는 단일사용자는 같은 우선권을 가지는 여러개의 스레드를 낳을수 있으며 체계를 부당하게 독점할수 있다.

10개의 처리기들을 가지는 사용자는 하나의 동등한 프로쎄스와 같은 우선권을 가지는 사용자보다 10배의 CPU시간을 받는다. 화페식일감짜기에 의해 체계관리자는 매 사용자에

게 전체 CPU시간(화페)의 한 몫을 배정한다.

처리기들보다 더 많은 프로쎄스들이 있으면 가장 높은 화페를 가지는 프로쎄스가 실행되도록 일정이 짜지며 CPU를 사용한다(즉 그것의 화페는 감소한다.).

더 낮은 화페를 가지는 프로쎄스들은 실행하지 않지만 자체의 화페는 유지한다. 마침내 기다리는 프로쎄스의 화페가 이전에 실행하는 프로쎄스의것을 릉가하여 실행되게 한다.

화페식일정짜기의 기본우월성은 일정짜기가 단기최량화를 하도록 하는 공정성의 장기보증이다. 요약하면 Cellular IRIX는 다음의 일정짜기기구들을 제공한다.

- **틀일정짜기** REACT/Pro틀일정짜기는 일정짜기를 완전히 담당할수 있으며 하나이상의 CPU들의 프로쎄스들을 정확한 전진속도를 담보하면서 신속하게 처리할수 있다.
- **무리일정짜기** Cellular IRIX는 병렬을 시작하기 위한 잠금들이나 신호기발들을 사용하여 호상통신하는 하나의 스레드묶음을 일정 짤수 있다. 이것은 어떤 스레드의 자물쇠보유가 그 자물쇠를 기다리고 있는 다른 스레드와 같은 시간간격으로 일정짜기가 진행되도록 한다.
- **처리기친화력** Cellular IRIX는 프로쎄스가 마지막으로 실행한 CPU를 자동적으로 표식하고 일부 자료나 코드가 캐쉬나 국부기억기에 남아 있다는 가정에 기초하여 같은 CPU우의 프로쎄스를 실행하려고 시도한다.

처리량지원−XFS파일체계

XFS는 높은 처리량과 담보된 I/O속도를 지원하는 64b실행기록파일체계이다. 500MB/s이상의 집체읽기/쓰기성능이 실현되였다.

높은 처리량은 령역개념과 실행기록기술에 의해 달성된다.

담보된 I/O속도는 응용들이 파일체계대역너비를 보존하도록 함으로써 달성된다.

XFS는 512~64KB의 범위의 론리블로크크기를 지원한다. 그것은 령역개념을 통합하여 하나의 령역에 포함된 자료가 디스크우의 련속적인 블로크들에 배치되도록 한다. XFS령역은 백만(2^{20})이상의 블로크들을 가질수 있다. 령역의 사용은 디스크탐색과 회전지연을 줄이게 하며 I/O처리량을 높인다.

실행기록(기록하기라고도 한다.)은 모든 파일체계변화들을 첨가용실행기록파일에 기록하는 기술이다. 이 파일은 후에 큰 덩어리들로 디스크에 써진다. 실행기록기술은 또한 체계회복시간을 줄이도록 한다. 체계는 체계파괴후 Unix파일체계들에서와 같이 시간을 소비하는 파일체계검사를 하지 않고 최근에 갱신된 파일체계트랜잭션의 작은 실행기록을 참조한다.

8. 5. 4. Origin 2000의 성능

Origin 2000의 구조적인 설계는 기억기와 I/O, 호상접속의 능력들이 모두 기계크기의 증대에 비례하여 확장되는 평형식체계로 되도록 하는데 큰 주의를 돌리고 있다.

이것은 표 8-7과 표 8-8에서 보여 주는바와 같이 하드웨어수준에서 크게 달성된다. Origin 2000의 박자주파수는 195MHz 또는 약 5ns의 주기시간이다.

이 표들로부터 집체대역너비(기억기와 I/O, 호상접속)는 처리기의 수에 따라 거의 선형으로 증가하며 지연은 단지 로그적으로 커진다는것을 알수 있다. 낮은 지연과 높은 대역너비는 Origin을 다른 DSM기계들보다 더 강하게 결합된 체계로 되게 한다. 초기성능평가는 Origin이 NPB나 SPLA와 같은 병렬과학계산성능평가기준들에서 좋은 속도증가를 달성한다는것을 보여 준다.

8. 6. CC - NUMA 구조들의 비교

표 8-9에서는 4개의 CC-NUMA구조들을 그것들의 마디구조와 내부마디의 접속, 캐쉬일관성규약들, 다른 성능개선특징들에 따라 비교한다.

매 체계는 CC-NUMA를 실현하는데서 서로 다른 방법들을 사용한다. Dash는 연구견본이며 다른것들은 상업용제품들이다.

표 8-7 Origin 2000의 계층기억기에서의 읽기지연

자료위치	읽기지연(박자)
등록기들	0
1차캐쉬(소편내장)	1~3
2차캐쉬(소편외장)	10
국부기억기	61
원격기억기(1개 경로조종기를 거친다.)	117
원격기억기(2개 경로조종기를 거친다.)	137
원격기억기(3개 경로조종기를 거친다.)	157

표 8-8 Origin 2000의 대역너비(GB/s)과 지연(ns)

체계구성 I/O수:마디수:CPU수	총 최대기억기 대역너비	총 최대I/O 대역너비	호상접속2등분 대역너비	기억기읽기 지연
1:1:2	0.78	1.56	-	313
2:2:4	1.56	3.12	1.56	497
2:4:8	3.12	6.24	3.12	601
4:8:16	6.24	12.48	6.24	703
8:16:32	12.48	24.96	12.48	805
16:32:64	24.96	51.2	12.48	908
32:64:128	49.92	99.84	24.96	1112

마디구조

대부분의 CC-NUMA체계들은 SMP마디들을 받아 들이고 있다. 여기서 하나의 마디는 모선을 통하여 마디의 국부기억기와 I/O부분체계에 접속된 여러개의 처리기들을 포함한다. 마디내의 캐쉬일관성은 snoopy규약에 의해 실시된다.

이것을 받아 들인 체계들에는 Stanford의 Dash, Sequent의 NUMA-Q, Data General의 NUMALiiNE와 HAL S1 등이 있다.

이 마디는 실현하기 쉽고 특히 Intel의 4개 처리기 SHV기판(8.1.3을 참고)과 같은 상품 SMP기판들의 유용성으로 하여 대중적이다. 그러나 CC-NUMA기계에서 snoopy모선식 SMP마디들의 사용은 다음과 같은 여러 부족점들을 가진다[393].

표 8-9 4가지 CC-NUMA구조들의 특징비교

구성방식특징	Stanford Dash	Sequent NUMA-Q	HP/Convex Exemplar	SGI/Cray Origin 2000
마디 구성방식	Snoopy모선을 가진 4-CPU SMP마디	Snoopy모선을 가진 4-CPU SMP마디	빗장을 가진 8-CPU SMP 마디	HUB를 가진 2-CPU non SMP마디
마디사이접속	2D그물	SCI고리	다중 2D SCI고리	살찐 초립방체
캐쉬일관성 규약	매 마디와 등록부에서 대역적인 Snoopy	SCI련결목록 일관성규약	SCI규약으로부터 수정	Dash규약으로부터 수정
다른 성능 특징	내부마디 캐쉬 대 캐쉬 공유	마디캐쉬, 무리일정짜기, 처리기친화력	마디캐쉬	무리일정짜기 페지이동, 배치, 복제

긴 원격기억기접근지연 snoopy모선은 여러 처리기의 지원을 요구하므로 높은 주파수를 달성하기 힘들다. 더우기 원격읽기는 국부snoopy모선과 호상접속, 원격snoopy모선을 통과하여야 최종적으로 원격기억기의 자료를 읽는다.

제한된 원격기억기대역너비 우의 원인들로 하여 원격기억기접근은 Dash의 경험이 보여 주는바와 같이 국부기억기대역너비의 절반 또는 1/3의 대역너비를 달성한다. 이 문제를 완화하기 위하여 Exemplar는 국부 및 원격기억기를 늘이는데 마디내의 크로스바를 사용한다. 그러나 이것은 원격접근지연을 많이 줄이지 못한다.

큰 입력자료체계비용 사용자들이 하나 또는 2개 처리기로 된 입력자료수준의 체계를 원할 때조차도 SMP마디의 전체 비용을 물어야 한다. 독자들에게 충고하지만 비용은 주로 기술에 의해서만 결정되지 않는다. 시장요구는 종종 가장 중요한 요인으로 된다. 값비싼 마디구조는 높은 용량으로 생산된다 해도(Intel의 SHV SMP기판들과 같이) 더 낮은 비용의 마디기술보다 더 값 눅게 될수 있다.

이 문제들을 극복하기 위하여 SGI2000는 비SMP마디구조를 채택한다. 매 마디가 2개

의 처리기를 가지지만 그것들은 SMP를 형성하지 않는다. 두개의 처리기들은 주소 및 자료모선들을 다중화하기 위하여 같은 물리모선을 사용하는 이전의 Intel 8088처리기와 같이 다중화된 방법으로 집선기(크로스바)에로의 하나의 모선을 공유한다.

목적은 집선기의 핀비용을 절약하는것이다. 집선기에로의 공유모선은 snoopy모선이 아니다. 마디내부의 캐쉬일관성은 등록부일관성하드웨어에 의하여 진행된다. 사실 2개 처리기로 된 Origin 2000마디의 기능들은 2개의 1개 처리기마디들과 같다.

현재의 실현에서는 비용을 줄이기 위하여 2개의 1개 처리기마디들을 하나의 단일마디로 묶는다(싼다.). 이때 더 좋은 원격접근지연과 대역너비를 달성한다.

원격기억기에로의 접근대역너비는 국부기억기대역너비와 거의 같다. 최선의 경우 국부읽기지연 대 원격읽기지연의 비는 약 2:1이다(표 8-7을 참고). 즉 원격기억기의 1개 hop에 접근하는것은 국부기억기보다 두배 더 길다. 지연은 hop의 수에 로그적으로 비례하여 커진다.

다른 체계들에서 국부 대 원격 지연비는 5:1~10:1이다.

체계호상접속

Dash는 2차원그물위상을 사용하고 있다.

NUMA-Q는 단일SCI고리를 사용하며 Exemplar는 여러 SCI고리들을 사용한다. Origin 2000는 굵은 하이퍼립방체위상을 사용한다. 더 큰 구성에 대하여 굵은 하이퍼립방체위상은 더 좋은 2등분대역너비를 제공하므로 다른 체계들보다 확대가능성이 더 높다. 그러나 오늘의 사용에서 대부분의 실제의 병렬콤퓨터들은 작은 구성을 가진다(실례로 20개이하의 처리기). 실제적인 설계, 생산, 시장조사를 통해서만 낮은 확대가능위상이 충분한가를 말할수 있다.

CC-NUMA기계들의 호상접속설계에서 공통적인 경향은 낮은 지연과 높은 대역너비를 달성하기 위하여 마디들을 매우 조밀하게 한데 묶을것을 요구하지 않는다는것이다.

조밀하게 한데 묶는것은 이전 세대의 MPP들과 PVP들의 기본특징이다. 이렇게 강하게 묶은 체계들에서 통신련결이나 케블들은 짧고 규정된 길이를 가져야 한다. 대비적으로 NUMA-Q에서 SCI고리의 련결들은 15m, Origin 2000의 Craylink는 5m만큼 길어도 된다.

일관성규약

현재의 모든 CC-NUMA체계들은 어떤 형태의 등록부식캐쉬일관성규약을 사용한다. 그러나 NUMA-Q와 Exemplar는 IEEE SCI표준에서 정의한것을 사용하며 Dash와 Origin 2000은 독자적인 규약들을 사용한다. Origin규약은 Dash규약으로부터 많이 수정된다. Exemplar규약은 IEEE SCI표준으로부터 약간 수정된다.

그밖의 특징들

하나의 SMP마디는 매개가 하나의 소편외장캐쉬(보통 2차캐쉬라고 부른다.)를 가지는 여러개의 처리기들을 가진다. SMP마디의 사용은 캐쉬들사이의 마디내의 자료공유를 가능하게 하는데 이것은 Dash구조로 실현된다. NUMA-Q와 Exemplar는 매 마디에 NUMA-Q에서는 원격캐쉬, Exempalr에서는 클라스터캐쉬라고 부르는 3차(third level)캐쉬를 사용한다.

이 마디캐쉬는 이전의 실패들로부터 그 마디로 가져 온 원격자료를 꺼내는데 사용된

다. 그러므로 마디는 원격자료를 가지며 자주 나가지 않아도 된다.

Dash와 Origin 2000은 마디캐쉬가 원격지연을 늘이고 대역너비를 줄인다고 보기때문에 그러한 마디캐쉬를 사용하지 않는다. 대신에 실패를 줄이기 위한 페지이동과 페지복제와 같은 기술들을 사용한다.

8. 7. 참고문헌주해와 련습문제

공유기억기다중처리의 문제점은 Alexander et al.[21], Amarasingbe et al.[28], Baron et al.[62], Catenzaro[132], Culler et al.[181], Dubois와 Scheurich[217], Gajski와 Peir[258], Hwang[327], lenoski와 Weber[406], Shang과 Hwang[552], Syanarayanan[534], Thakkar et al.[610], Yew와 Wah[662]에서 론의된다.

Holt et al.[319]는 DSM기계의 응용들과 구조적병목들을 해석한다.

Hwang et al.[329]는 과학적인 병렬처리를 위한 수직다중처리기구조를 제기한다. 다중처리기들에 대한 성능모형은 Marson et al.[430], Bhuyan과 Zhang[8]에서 론의된다. 부분계타방정식들을 풀기 위한 다중처리기들의 사용은 Wang과 Hwang[635]에서 론의된다.

Intel SHV기판들은 Intel Web Sites[346]에서 론의된다. SHV기판들은 Clark와 Alnes[159], Lovett와 Clapp[420], Weber et al.[639]에서 서술된바와 같이 여러 상업용공유기억기체계들에서 사용되였다.

Sun SMP봉사기들은 Catenzaro[132], Ceklevet al.[134], Fenwick et al.[239]에서 서술된다.

Gigaplane은 Singhal et al[561]에서 서술되는 Sun Ultra Enterprise 10000에서 사용되였다.

HP/Convex Exempla는 Convex[165, 166], Brewer와 Astfak[110], Sterling et al[586]에서 서술된다.

Sequent NUMA-Q 2000은 Lovett Clapp[420], Lovett et ak[421]에서 서술된다.

SGI Origin 2000체계는 Laudon과 Lenoski[393], SGI[550], Whitney et al.[643]에서 론의된다.

문 제

문제 8.1. 다음의 용어들을 정의하시오.

(1) 분산트랜잭션모선

(2) 특수기억기요청

(3) 원격캐쉬(마디캐쉬, 클라스터캐쉬로도 알려 져 있다.)

(4) 캐쉬행의 home마디

문제 8.2. 8.1.3의 INTEL SHV SMP기판을 고찰하시오.

(1) OEM포구란 무엇인가?

(2) OEM포구를 리용할수 있는것 3가지를 드시오.

(3) 32b의 캐쉬행크기와 60ns의 주기억기주기시간을 가진다고 하자. 기판에 하나의 처리기만이 있을 때 최소기억기접근지연을 구하시오.

(4) 32b의 캐쉬행크기와 60ns의 주기억기주기시간을 가진다고 하자. 기판에 4개의 처리기가 있을 때 달성할수 있는 최대기억기대역너비를 구하시오.

문제 8.3. 8.4에서 Sequent NUMA-Q 2000체계를 고찰하시오.

(1) NUMA-Q설계의 기본전략은 상품 및 표준구성요소들을 사용하는것이다. NUMA-Q에서 사용된 5개의 상품구성요소들과 2개의 전용구성요소들을 드시오.

(2) 왜 하나의 마디에 2개의 원격캐쉬지령(꼬리표, tag) SDRAM이 있는가?

문제 8.4. 그림 8-17을 참고하면서 다음의 물음에 대답하시오.

(1) 모든 express련결들을 첨부하여 그림 8-17을 다시 그리시오.

(2) 모든 실패련결들을 첨부하여 그림 8-17을 다시 그리시오.

(3) 128개 마디(256개 처리기) Origin 2000의 굵은 하이퍼립방체위상을 설명하시오.

(4) 512개 마디(1024개 처리기) Origin 2000의 굵은 하이퍼립방체위상을 설명하시오.

문제 8.5. Sequent NUMA-Q 2000과 SGI Origin 2000의 구조적차이점을 5개 드시오. 기계크기, 기억기용량과 같은 능력에서의 차이는 없다. 매 구조적차이에 대하여 어느 체계가 우월한가를 설명하시오.

문제 8.6. 이 장에서 서술한 적어도 하나의 체계에 의해 자료국부성을 실현하는데 사용된 기술 4가지를 들고 간단히 설명하시오.

문제 8.7. 이 장에서 서술한 적어도 하나의 체계에 의해 지연을 숨기는데 사용된 기술 4가지를 들고 간단히 설명하시오.

문제 8.8. 이 장에서 서술한 적어도 하나의 체계에 의하여 지원확대가능성을 높이는데 사용된 기술 5가지를 들고 간단히 설명하시오.

문제 8.9. 이 장에서 서술한 적어도 하나의 체계에 의하여 유용성을 높이는데 사용된 기술 5가지를 들고 간단히 설명하시오.

문제 8.10. 이 장에서 서술한 적어도 하나의 체계에 의하여 관리가능성을 높이는데 사용된 기술 4가지를 들고 간단히 설명하시오.

제9장. 클라스터화와 유용성의 지원

클라스터들은 콤퓨터의 클라스터, 워크스테이션의 클라스터, 워크스테이션의 망 등과 같은 많은 이름으로 콤퓨터전문가들에게 알려 져 있다. 클라스터화는 비용이 효과적이고 확장가능한 병렬콤퓨터들을 개발하는데서 하나의 동향으로 되고 있다. 클라스터의 기본개념들은 1.4에서 소개되였다.

이 장에서는 클라스터설계원리들을 론의하며 여러대의 콤퓨터들을 클라스터화하기 위하여 필요한 하드웨어 및 쏘프트웨어의 지원을 고찰한다.

다음장에서는 이 클라스터화기술들이 콤퓨터들의 상업 또는 연구클라스터들에 어떻게 통합되는가를 보여 준다.

9. 1. 클라스터화에서의 도전

먼저 클라스터들에 대한 분류법을 줌으로써 클라스터개념을 명백히 한다.

다음 임의의 클라스터체계의 설계자와 사용자가 처리하여야 할 문제들을 명백히 한다. 이 문제들을 풀기 위한 기술들은 순차적인 부분들에서 론의된다.

9. 1. 1. 클라스터의 분류

클라스터는 문헌들에서 종종 동의어들을 혼동시키면서 여러가지 방법으로 분류되였다. 4개의 독립적인 속성들 즉 묶기와 조종, 동질, 보안을 사용하여 클라스터들을 분류한다. 기본흐름의 유지 및 간단화를 위하여 표 9-1에서와 같이 매 속성에 대하여 2개의 값만을 허용한다. 현재 전용클라스터와 기업클라스터라고 하는 2개의 조합만이 실제의 클라스터들에서 사용된다. 그러나 개념적으로는 모두 16개의 조합이 가능하다.

표 9-1 클라스터분류에서 사용한 속성들

속성들	속성값	
묶기	치밀하다	늘어 지다
조종	집중식	분산식
동질	동질	이질
보안	밀폐식	로출식
실례	전용클라스터	기업클라스터

묶기

클라스터의 마디들을 치밀하게 묶을수 있다. 치밀한 클라스터에서 마디들은 방안에서 있는 하나 또는 그이상의 시렁에서 치밀하게 묶여 지며 주변장치들(모니터, 건반, 마우스 등)에 련결되지 않는다.

Goden Bell은 그러한 마디들을 머리 없는 **워크스테이션**이라고 부른다[67]. 클라스터에서 마디들은 그것들의 상용주변장치들(즉 그것들은 완전한 SMP들과 워크스테이션들, PC들이다.)에 련결되며 서로 다른 방들, 서로 다른 건물들 지어 지리적으로 멀리 떨어 진 지역들에 위치해도 된다.

묶기는 통신선길이에 직접적으로 영향을 주며 따라서 호상접속기술의 선택에도 영향을 준다. 치밀한 클라스터는 흔히 전용으로 된 높은 대역의 낮은 지연통신망을 리용할수 있으며 한편 늘어 진 클라스터의 마디들은 보통 표준LAN들이나 WAN들을 통하여 접속된다.

조종

클라스터는 집중식 또는 분산식으로 조종되거나 관리될수 있다. 치밀한 클라스터는 보통 집중식조종을 가지며 늘어 진 클라스터는 두 형식으로 조종될수 있다.

집중식클라스터에서 모든 마디들은 중앙운영자에 의하여 소유되고 조종되며 관리되고 지배된다.

분산식클라스터에서 마디들은 개별적인 소유자들을 가진다. 실례로 백화점에서 호상접속된 탁상용워크스테이션들로 이루어 진 클라스터를 보면 매 워크스테이션은 한명의 종업원에게 소유된다.

소유자는 임의의 시각에 워크스테이션을 재설정하거나 갱신하며 지어 완료시킬수도 있다. 단일조종점이 없으면 그러한 클라스터에 대한 체계관리가 대단히 힘들게 된다. 이것은 또한 프로쎄스의 일정짜기, 작업부하의 이동, 검사점, 회계 등을 위한 특수한 기술들을 요구한다.

동질(Homogeneity)

동질의 클라스터란 마디들이 같은 가동환경을 채용한다는것을 의미한다. 즉 그것들은 같은 처리기구조와 같은 조작체계를 사용한다. 흔히 마디들은 같은 판매자들로부터 구입된다.

이질클라스터는 서로 다른 가동환경의 마디들을 사용한다. 이질클라스터들에서 호상운영성은 중요한 문제이다. 실례로 프로쎄스의 이동은 흔히 부하평형이나 유용성에 필요하다.

동질클라스터에서 프로쎄스의 2진코드는 다른 마디로 옮겨 가 실행을 계속할수 있다. 이질클라스터에서는 프로쎄스가 다른 가동환경의 마디로 이동하면 2진코드는 실행할수 없기때문에 동질클라스터에서처럼 다른마디로 옮겨 가 실행할수 없다.

보안

클라스터내부의 통신은 로출되거나 밀페될수 있다.

로출된 클라스터에서 마디들사이의 통신경로들은 바깥세계에 로출된다. 바깥쪽의 기계는 표준규약들(즉 TCP/IP)을 사용하여 통신경로들에 접근할수 있으며 따라서 개별적인 마디들에도 접근할수 있다. 그러한 로출된 클라스터들은 실현하기는 쉽지만 여러가지 부족점을 가진다. 로출된 클라스터내의 통신은 통신부분체계가 보안을 담보하는 보충적인 동작을 수행하지 않는다면 안전하지 못하다.

바깥쪽 통신들은 클라스터내의 통신들을 예측할수 없는 형태로 와해시킬수 있다. 실례로 무거운 BBS통신흐름이 생산일감들을 와해시킨다. 표준통신규약들은 높은 부가처리를 가지는 경향이 있다.

밀페식클라스터에서 클라스터내의 통신은 바깥세계로부터 차페되며 이것은 우의 문제들을 완화시킨다. 부족점은 현재 효률적이고 밀페된 클라스터내부의 통신을 위한 표준이 없다는것이다. 결과 대부분의 상업 또는 대학의 클라스터들은 어떤 종류의 규약들중의 하나를 통하여 고속통신을 실현한다.

전용 대 기업 클라스터

Louis Turcotte[620]은 클라스터를 2개의 부류 즉 전용클라스터와 기업클라스터로 나누었다. 전용클라스터는 다음의 특징들을 가진다.

- 전형적으로 중앙콤퓨터실의 책상면시렁에 설치된다.
- 전형적으로 같은 형태의 마디들에 의하여 동질적으로 구성된다.
- 대형콤퓨터와 같이 단일한 관리자그룹에 의해 관리된다.
- 전형적으로 앞단의 체계를 통하여 접근된다.
- 전용클라스터는 전통적인 대형콤퓨터들이나 초고속콤퓨터들의 내용으로 사용된다.

전용클라스터는 단일한 하나의 기계로서 설치되고 관리된다. 많은 사용자들이 클라스터에 접속하여 호상작용 및 묶음일감들을 실행할수 있다. 클라스터는 감소된 응답시간만큼 더 개선된 처리량을 제공한다.

기업클라스터는 주로 마디의 아무 일도 하지 않는 자원들을 리용하는데 사용된다. 이것은 다음의 특징들을 가진다.

- 매 마디는 보통 필요한 모든 주변장치들이 련결된 충분히 완성된 SMP나 워크스테이션, PC이다.
- 마디들은 전형적으로는 지리적으로 분산되여 있으며 반드시 같은 방에 또는 같은 건물에 있어야 하는것은 아니다.
- 마디들은 개별적으로 여러 소유자들에게 소유된다. 클라스터관리자는 마디가 그 사용자에 의해 임의의 시각에 막힐수 있기때문에 마디들에 대한 조종을 제한하기만 한다. 소유자의 론리적일감들은 기업일감들보다 우선권이 더 높다.
- 클라스터는 종종 이질의 콤퓨터마디들로 구성된다. 마디들은 흔히 낮은 가격의 이써네트를 통하여 접속된다.

9. 1. 2. 클라스터의 구조

클라스터의 마디들을 그림 1.2에서 보여 주고 그림 9-1에 다시 그린것과 같은 3가지 방법들중의 한가지 방법으로 접속될수 있다.

대부분의 클라스터들에서는 공유 없는 구조가 사용된다. 여기서 마디들은 I/O모선을 통하여 접속된다. 디스크공유구조는 업무응용들에서 작은 규모의 유용성클라스터를 허용한다. 한 마디가 고장나면 다른 마디가 계승한다.

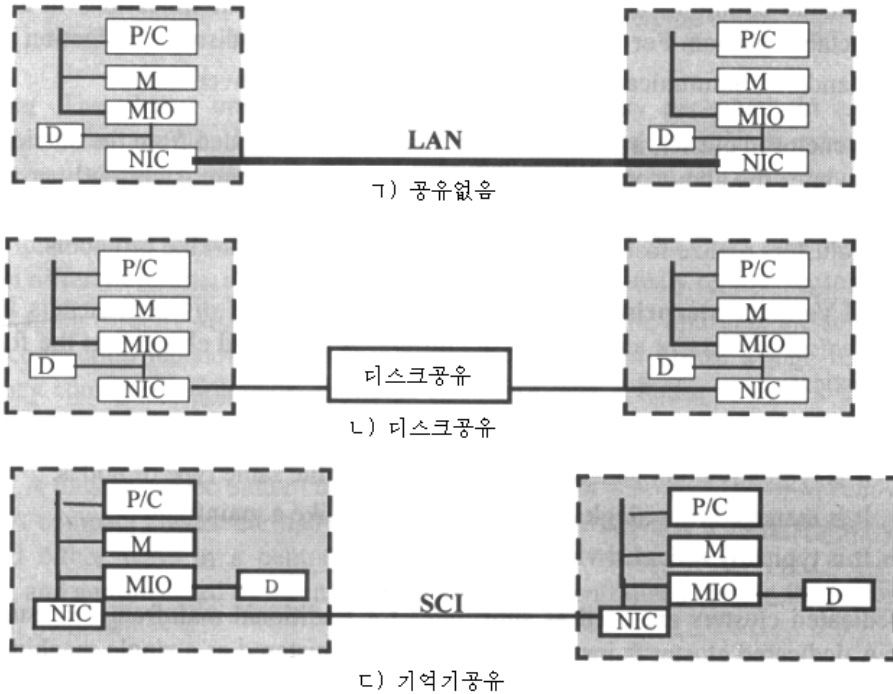



그림 9-1. 클라스터의 마디들을 접속하는 3가지 방법

기억기공유클라스터들은 최근에 생겨 났다. 한가지 실례는 Sequent NUMA-Q콤퓨터 [345]이다. 이것은 하나의 마디에서 4개의 Intel Pentium Pro처리기를 사용하며 마디들은 SCI고리로 접속된다. SCI는 NIC모듈을 통하여 마디의 기억기모선에 접속된다. 기억기공유구조에서 호상접속은 매 마디의 기억기모선에 련결된다. 다른 2가지 구조에서는 호상접속이 I/O모선에 련결된다. 기억기공유구조는 다음의 원인으로 하여 다른것들보다 실현하기 힘들다.

- 기억기모선이 보통 I/O모선보다 더 높은 주파수에서 동작한다.
- 기억기모선에 대한 표준을 폭 넓게 받아 들이지 않는다. 그러나 I/O모선들에 대한 그러한 표준들은 있다. 최근에 인기 있는 표준은 PCI I/O모선표준이다. 그러므로 만일 더 빠른 이써네트를 클라스터마디의 PCI모선에 련결하는 NIC기판을 실현하면 이 기판이 I/O모선으로서 PCI를 사용하는 다른 체계들에서 사용될수 있다는것을 담보할수 있다.
- I/O모선은 기억기모선보다 더 느린 속도로 동작한다. PCI모선으로 접속된 클라스터를 생각하자. 처리기들이 갱신되거나 마디구조가 다른 가동환경으로 변화될 때 새 체계가 여전히 PCI를 사용하는 한 호상접속과 NIC는 변화시키지 않아도 된다. 기억기공유클라스터에서 처리기의 변화는 마디기판과 NIC기판의 재설계를 의미한다.

9.1.3. 클라스터의 설계문제점

클라스터의 개발과 리용에서는 여러개의 문제들이 고려되여야 한다. 이것들은 많은 연구가 진행되였지만 여전히 활발한 연구 및 개발분야이다.

4개의 중요한 문제들 즉 유용성지원, 단일체계영상, 일감관리, 효률적인 통신을 아래에서 소개한다.

유용성지원

클라스터들은 처리기들과 기억기들, 디스크들, I/O장치들, 망들, 조작체계형태들에서 많은 여유를 가지는 가격이 적당히 높은 유용성을 제공할수 있다. 그러나 이 능력을 실현하려면 유용성기술들이 필요하다. 이 기술들은 DEC클라스터들(10.4)과 IBM SP2(10.3)이 높은 유용성을 어떻게 달성하는가를 론의할 때 설명한다.

단일체계영상

이써네트로 접속한 워크스테이션들의 모임이 반드시 클라스터인것은 아니다. 클라스터는 단일체계이다.

간단한 실례를 들자. 워크스테이션이 300Mflop/s의 처리기와 512MB의 기억기, 4GB의 디스크를 가지며 50명의 활동사용자들과 1000개의 프로쎄스들을 지원할수 있다고 하자. 그러한 워크스테이션들 100개를 클라스터화하여 30Gflop/s의 처리기와 50GB의 기억기, 400GB의 디스크를 가지며 5000명의 활동사용자들과 100000개의 프로쎄스들을 지원할수 있는 하나의 거대한 워크스테이션과 동등한 단일체계를 만들수 있겠는가?

이것은 흥미 있는 목표이지만 실현하기 대단히 힘들다. 단일체계영상(SSI)은 이 목표를 달성하는데로 지향되고 있다.

일감관리

클라스터들은 보통 고도로 리용되지 않는 일반적인 워크스테이션이나 PC마디들을 벗어 나 높은 체계리용을 달성한다. 일감관리쏘프트웨어는 묶음처리와 부하평형, 병렬처리, 다른 기능을 제공하는데 필요하다. 이 문제는 9.5에서 론의된다.

효률적인 통신

MPP보다도 클라스터를 위한 효률적인 통신부분체계를 개발하는것이 힘들다. 여기에는 여러가지 리유가 있다.

- 더 높은 마디복잡성으로 하여 클라스터마디들은 MPP마디들만큼 치밀하게 묶을수 있다.
- 클라스터내에서 마디호상간의 물리적선길이는 MPP에서보다 더 길다. 이것은 집중식클라스터들에서도 마찬가지이다. 긴 선은 더 큰 호상접속망지연을 가져 온다. 그러나 보다 중요하게 더 긴 선은 더 높은 믿음성과 박자이즈러짐, 교차대화문제들을 가진다. 이 문제들은 믿음직하고 안전한 통신규약을 요구하며 부가처

리를 증가시킨다.

- 클라스터들은 흔히 TCP/IP와 같은 표준통신규약들을 가지는 상품망들(실례로 이써네트, ATM)을 사용한다. 상품적구성요소들은 Moor의 법칙을 만족시키지만 TCP/IP규약들은 높은 부가처리를 가진다. 저수준통신규약들은 7.4.3에서 론의한 것과 같이 더 효률적이다. 그러나 현재 저수준통신규약에 대한 표준은 없다.

7.4.3에서 론의한 통신기술은 클라스터들에서 사용될수 있다.

효률적인 통신이 어떻게 달성되는가를 Berkeley NOW과제(10.5)와 DEC클라스터들(10.4), IBM SP2(10.3)에서 보여 준다.

리상적인 클라스터구조

여기서는 그림 9-2에서 보여 주는것과 같은 구조를 가지는 리상적인 클라스터를 상상한다. 프로세스마디들은 워크스테이션과 PC, SMP봉사기, 지어 초고속콤퓨터들이다. 마디의 조작체계들은 다중사용자, 다중과제, 다중스레드체계들이다. 마디는 동질이거나 이질일수 있다. 마디들은 하나 또는 그이상의 고속상품망들에 의해 호상접속된다. 이 망들은 표준적인 통신규약들을 사용하며 이써네트우의 현재의 TCP/IP보다 두자리 더 높은 속도로 동작한다.

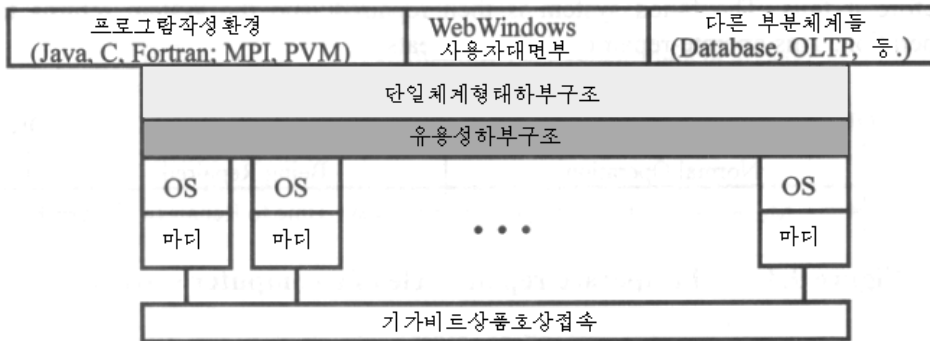



그림 9-2. 유용성과 단일체계영상에 대한 충분한 지원을
가지는 작업클라스터의 구조

매 마디의 망대면부회로는 표준I/O모선(실례로 PCI)에 접속된다. 처리기나 조작체계가 변화되면 구동쏘프트웨어만 변화시키며 망이나 망대면부는 변화시키지 않는다.

가동환경독립의 쏘프트웨어부분체계들의 모임은 마디의 가동환경의 최상위에 위치한다.

높은 유용성봉사들을 제공하는 하나의 유용성부분체계가 있다.

단일체계영상층은 단일입력자료, 단일파일계층, 단일조종점, 단일일감관리체계를 제공한다.

단일기억기는 콤파일러 또는 실시간서고기술의 도움으로 실현되게 된다.

단일프로세스의 공간은 반드시 지원되지 않는다.

클라스터의 이러한 유용성은 3가지 형태의 부분체계들에 의해 지원된다.

- 첫째로, 일반적인 자료기지들과 OLTP감시기들은 사용자에게 워크스테이션에서와 꼭 같은 환경을 제공한다.
- 둘째로, 일반적인 순차프로그람작성환경외에 클라스터는 표준적인 언어들과 통신서고들에 기초한 병렬프로그람작성을 지원한다. 환경은 또한 결함수정, 감시 등을 위한 도구들도 포함한다.
- 셋째로, 사용자대면부부분체계는 Web대면부와 윈도우즈 GUI의 우월성들을 결합한다. Foxetal[252]는 그러한 환경을 WebWindows라고 부른다. 그것은 또한 사용자들이 도움말과 개별지도, 실례, 론증, 자주 제기되는 물음들에 대한 대답 등을 쉽게 참고할수 있도록 여러가지 프로그람작성환경과 일감관리도구, 초본문 및 탐색지원에 대한 친절한 사용자련결을 제공한다.

9. 2. 클라스터화에 대한 유용성지원

건전하고 고도로 유용한 체계설계에서는 흔히 3가지 술어 즉 믿음성, 유용성, 편리성(RAS로 생략한다.)이 함께 나타난다.

유용성은 아래에서 정의하는바와 같이 믿음성과 봉사성의 결합개념이기때문에 가장 흥미 있다.

믿음성은 체계가 얼마나 오래동안 고장없이 동작할수 있는가를 측정한다.

유용성은 체계가 사용자에게 리용가능한 시간의 백분률 즉 체계실행시간의 백분률을 가리킨다.

편리성은 하드웨어와 쏘프트웨어의 정비, 수리, 갱신 등을 포함하는 체계봉사가 얼마나 쉬운가로 평가한다.

실례 9.1. 표준(average)콤퓨터체계의 유용성

RAS에 대한 요구는 실제적인 시장요구에 따른다.

최근의 Find/SVP개괄은 큰 1000개 회사들속에서 다음의 특징들을 발견하였다. 평균 4시간의 정지시간을 가지는 표준콤퓨터는 한해에 9배씩 적어 진다. 정지시간당 평균소득손실은 82500$이다. 그러한 무거운 벌금으로 하여 많은 회사들은 24×365의 유용성을 제공하는 체계 즉 하루에 24시간, 1년에 365일 리용가능한 체계를 위하여 노력하고 있다.

9. 2. 1. 유용성개념

그림 9-3에서와 같이 콤퓨터체계는 보통 그것이 고장나기전의 시간구간에서 동작한다. 다음 고장난 체계는 수리되여 정상적인 동작에로 돌아 온다.

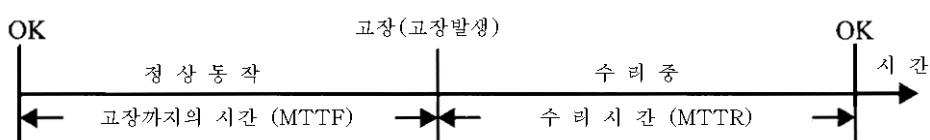



그림 9-3. 콤퓨터체계의 동작-수리주기

믿음성과 유용성, 편리성(RAS)에 대한 정의와 척도들이 혼돈되는 경우들도 있다. 여기서는 이 술어들을 다음과 같이 구분한다.

정의 9.1. 체계의 믿음성은 고장나기까지의 평균시간(mean time to failure, MTTF)으로 측정된다. 이것은 체계(또는 체계의 구성요소)가 고장나기전까지 정상동작의 평균시간이다. 편리성에 대한 척도는 수리하기까지의 평균시간(mean time to repare, MTTR)이다. 이것은 체계를 수리하고 고장나기전의 작업환경을 재현하는데 걸리는 평균시간이다. 그러면 체계의 유용성은

$$\text{유용성} = \frac{MTTF}{MTTF + MTTR} \tag{9.1}$$

로 정의된다.

예정고장 대 비예정고장

RAS를 고찰할 때 체계의 정상동작을 방해하는 임의의 사건을 **고장**이라고 부른다. 이것은 다음의것들을 포함한다.

- **비예정고장** 체계는 조작체계파괴, 하드웨어고장, 망의 비접속, 사람의 동작오유, 정전 등으로 하여 파괴된다. 이것들은 모두 간단히 고장이라고 부른다. 체계는 수리되여 그 고장을 고쳐야 한다.
- **예정된 닫기** 체계는 파괴되지 않지만 갱신과 재설정, 정비를 위하여 주기적으로 정상동작을 중지한다. 체계는 또한 주말이나 명절에 닫길수 있다.

표 9-2 콤퓨터체계형태들의 유용성

체계형태	유용성(%)	한해의 정지시간
전통적인 워크스테이션	99	3.6일
고유용성체계	99.9	8.5시간
고장회복체계	99.99	1시간
고장허용체계	99.999	5분

표 9-2는 대표적인 몇개 체계들의 유용성값들을 보여 준다. 실례로 워크스테이션은 99%의 유용성을 가지는데 이것은 한해의 99%시간동안 동작하고 있다는것을 의미한다. 정지시간은 1년의 1%인데 이것은 거의 3.5일이다.

유용성의 정의는 예정된 정지시간을 고려하지 않는데 이것이 중요할수도 있다. 실례로 많은 초고속콤퓨터의 설치는 주당 여러 시간의 예정된 정지시간을 가지지만 전화교환기체계는 해마다 몇분의 정지시간(예정된 또는 예정되지 않은)도 허용할수 없다.

일시적인 고장 대 영구적인 고장

많은 고장들은 일시적이다. 그것들은 일시적으로 생기고 사라진다. 그것들은 어떤 구성요소의 교체도 없이 처리될수 있다. 표준적인 방법은 체계를 복구하고 시동하는것이다.

실례로 굳어 진 건반과 창문과 같은 일시적인 고장들을 회복하기 위하여 PC를 다시 초기적재한다.

영구적인 고장은 다시 초기적재하여 고칠수 없다. 일부 하드웨어 또는 쏘프트웨어의 구성요소는 수리되거나 교체되여야 한다. 실례로 재초기적재는 체계의 하드디스크가 파괴되면 수행되지 않는다.

부분적고장 대 전체적고장

전체 체계를 리용할수 없게 하는 고장들을 **전체적고장**이라고 부른다.

체계의 부분에만 영향을 주는 고장은 만일 체계가 낮아 진 능력으로라도 여전히 리용가능하면 **부분적고장**이라고 부른다.

유용성을 높이는 주요한 방법은 단일고장점들을 체계적으로 제거하여 가능한껏 많은 고장들을 부분고장으로 만드는것이다.

여기서 단일고장점들은 전체 체계를 허무는 고장을 가져온 하드웨어 또는 쏘프트웨어의 구성요소들이다.

실례 9.2. STP와 콤퓨터들의 클라스터에서 단일고장점들

그림 9-4 ㄱ)에서 보여 주는 SMP에서 공유기억기와 OS형태, 기억기모선은 모두 단일고장점들이다.

한편 처리기들은 단일고장점이 아니다.

이써네트로 접속한 워크스테이션들의 클라스터(그림 9-2 ㄴ)에는 여러개의 OS형태가 있는데 그 매개는 하나의 워크스테이션안에 상주한다. 이것은 SMP의 경우와 같이 OS에 의한 단일고장점으로 되지 않는다. 그러나 이써네트는 하나의 단일고장점으로 되는데 이것은 그림 9-4 ㄷ)에서 제거된다. 여기서 두개의 통신경로를 제공하기 위하여 고속망이 보충된다.

그림 9-4 ㄴ)와 그람 9.4 ㄷ)의 클라스터에서 한마디가 고장나면 마디의 응용들모두가 고장날뿐아니라 마디가 수리될 때까지 마디의 모든 자료는 사용될수 없다.

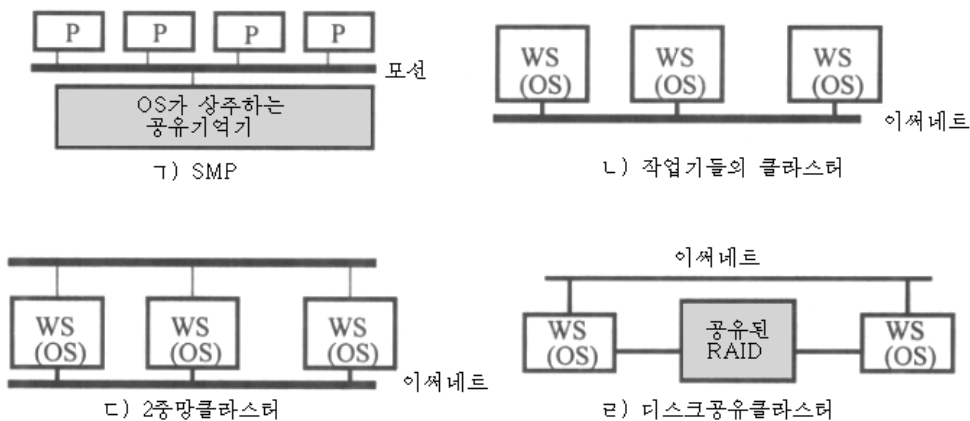



그림 9-4. SMP와 3개의 클라스터의 단일고장점들

그림 9-4 ㄹ)의 공유디스크클라스터는 한가지 구제수단을 제공한다. 체계는 지속적인 자료를 공유디스크에 기억할수 있으며 검사점응용들은 중간결과들을 디스크에 보존할수 있다. 따라서 하나의 마디가 고장날 때 자료는 잃어 지지 않는다.

실례 9.3. 콤퓨터들의 클라스터의 유용성

그림 9-4 ㄹ)의 클라스터를 고찰하자.

하나의 마디만이 고장날수 있다고 가정하자. 체계의 나머지(실례로 호상접속과 공유 RAID디스크)는 100% 리용가능하다. 또한 한 마디가 고장날 때 그것의 작업부하는 다른 마디로 0시간에 절환된다.

(1) 예정정지시간을 무시할 때 클라스터의 유용성은 얼마인가?
(2) 클라스터가 주당 1시간의 정비를 요구할 때 유용성은 얼마인가?
(3) 한번에 한마디씩 주당 1시간의 정비가 필요하다면 유용성은 얼마인가?

해답:

(1) 표 9-2로부터 워크스테이션은 99%의 시간동안 리용가능하다. 두개의 마디가 정지되는 시간은 $0.01\times0.01=0.0001$ 또는 0.01%이다. 그러므로 유용성은 99.99%이다. 따라서 클라스터는 해마다 단지 1시간의 정지시간을 가지는 고장회복체계이다.
(2) 예정된 정지시간은 한해에 52시간 즉 $52/(365\times24)=0.0059$이다. 따라서 전체 정지시간은 $0.59\%+0.01\%=0.6\%$이다. 클라스터의 유용성은 99.4%로 된다.
(3) 다른 마디는 고장나고 한 마디는 정비되고 있는것과 같은 가망 없는 정황을 무시한다고 하자. 그러면 유용성은 99.99%로서 (1)에서와 같다.

9．2．2．유용성기술

식 (9.1)로부터 체계의 유용성을 높이는데는 기본적으로 2가지 방법 즉 MTTF를 늘이는것과 MTTR를 줄이는것이 있다. 결국 MTTF의 증가는 체계유용성의 증가로 된다.

콤퓨터공업은 믿음직한 체계를 만들기 위하여 노력하여 왔다. 오늘의 워크스테이션들은 수백~수천시간의 범위에서 MTTF를 가지고 있다. 그러나 MTTF를 더욱 개선하는것은 매우 힘들며 비용이 많이 든다.

클라스터들은 체계의 MTTR를 줄이는데 기초한 높은 유용성해결책을 제공한다. 다중마디클라스터는 워크스테이션보다 더 낮은 MTTF(따라서 더 낮은 믿음성)를 가지며 고장이 더 자주 일어 난다. 그러나 고장들은 더 높은 유용성을 달성하기 위하여 빨리 회복된다. 클라스터에서의 여러가지 유용성기술들을 아래에서 론의한다.

분리된 여유

임의의 체계에서 유용성을 높이는 주요한 기술은 여유구성요소들을 사용하는것이다. 하나의 구성요소(기본구성요소)가 고장나면 그것이 제공하던 봉사는 다른 구성요소(복제구성요소)가 이어 받는다. 더우기 기본 및 복제구성요소들이 호상 분리된다면 그것들은 같은 고장원인의 영향을 받지 않는다.

클라스터들은 전원공급과 선풍기, 처리기, 기억기, 디스크, I/0장치, 망, 조작체계형태 등에서 여유를 가지는 높은 유용성을 제공한다. 주의 깊게 설계된 클라스터에서 여유는 역시 분리된다.

분리된 여유는 여러가지 우점을 준다.

첫째로, 분리된 여유를 가지고 설계된 구성요소는 단일고장점이 아니며 구성요소의 고장은 전체 체계의 고장을 일으키지 않는다.

둘째로, 고장난 구성요소는 체계의 나머지가 여전히 동작하는 상태에서 수리될수 있다.

셋째로, 기본 및 복제구성요소들은 호상 검사할수 있으며 호상 결함을 찾을수 있다.

IMB SP2의 통신부분체계는 여유분리설계의 좋은 실례이다. 모든 마디들은 2개의 망 즉 이써네트와 고성능교환기(HPS)로 접속된다. 매 마디는 이 망들에 접속하기 위한 2개의 분리된 대면부기판을 사용한다. 2개의 통신규약 즉 표준 IP와 사용자공간(US)규약이 있으며 매 규약은 각각의 망에서 실행할수 있다. 망이나 규약이 고장나면 다른 망이나 규약이 이어 받을수 있다.

실례 9.4. 쏘프트웨어의 믿음성을 높이는 N판본프로그람작성

사명이 중요한 쏘프트웨어체계를 구성하는데서 공통적인 여유분리방법을 N판본프로그람작성(NVP)이라고 부른다. 쏘프트웨어는 다른것들이 있다는것조차도 모르는 N개의 분리된 팀들에 의해 실현된다. 서로 다른 팀들은 서로 다른 알고리듬, 프로그람작성언어, 환경도구 지어 서로 다른 가동환경을 사용하여 쏘프트웨어를 실현하게 한다.

고장허용체계에서 N개의 판본들은 모두 동시에 실행되며 그 결과들은 언제나 비교된다. 결과들이 다르면 체계는 고장이 생겼다는것을 알게 된다. 그러나 분리된 여유로 하여 그 고장이 N개 판본들 대부분이 동시에 고장나도록 하는 일은 거의나 일어 나지 않는다. 그리하여 체계는 대부분의 판본들의 투표로 생성된 정확한 결과를 가지고 작업을 계속한다.

널리 리용되는 사명이 덜 중요한 체계에서는 한번에 하나의 판본만을 실행해야 한다. 매 판본은 내장된 자체검사능력을 가진다. 한 판본이 고장나면 다른 판본이 이어 받을수 있다.

실례 9.5. IBM HACMP에서 단일고장의 제거

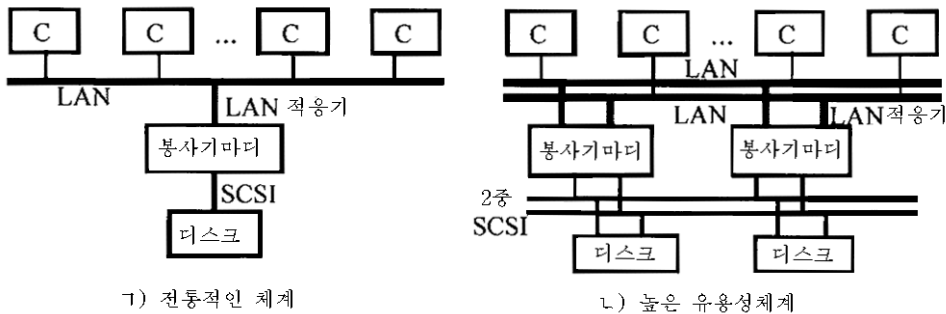



그림 9-5. IBM HACMP에서 모든 단일고장점을 제거하기 위한 클라스터자원들이 복제

IBM HACMP(high-availability cluster multiprocessing)체계의 설계는 그림 9-5에서 보여 주는바와 같이 모든 단일고장점들을 조심스럽게 제거한다.

전통적인 의뢰기-봉사기구성은 그림 9-5 ㄱ)에서 보여 준다. 여기서 여러 의뢰기들(ㄷ)은 봉사기마디가 이써네트와 같은 LAN을 통하여 제공하는 봉사에 접근한다. 봉사기마디는 SCSI모선을 통하여 외부의 디스크에 접속된다. 이 전통적인 체계는 5개의 가능한 단일고장점을 가진다.

(1) LAN망
(2) 봉사기마디의 LAN적응기(굵은 선으로 보여 준다.)
(3) 봉사기자체
(4) SCSI모선
(5) 외부디스크

그림 9-5 ㄴ)의 높은 유용성체계는 여분설계를 사용하여 모든 단일고장점들을 제거한다. 여분은 우에서 렬거한 5개의 체계자원들 모두에 대한 복제로부터 생긴다. 즉 2개의 LAN, 두개의 봉사기마디, 봉사기마디당 두개의 LAN적응기기판, 2조의 SCSI모선, 2개의 디스크렬이 사용된다.

이것은 높은 유용성을 달성하는데 대단히 값이 비싼 방법이지만 IBM HACMP는 고객의 요구에 따라 이 방법을 받아 들이였다.

여분의 구성요소들은 서로 다른 가격과 유용성, 성능요구에 따라 서로 다른 방법으로 구성될수 있다. 다음의 3가지 구성이 자주 사용된다.

- **활성대기** 기본구성요소는 봉사를 제공한다. 한편 여분의 복제구성요소는 어떤 작업도 하지 않고 대기하다가 기본구성요소가 고장나자마자 이어 받기 위하여 준비되여 있다. 보다 경제적인 설계는 여러 기본구성요소들을 복제하는데 하나의 대기구성요소를 사용한다. 복제구성요소는 아무것도 하지 않기때문에 고장난 기본구성요소를 빨리 이어 받을수 있다.
- **호상 이어 받기** 모든 구성요소들은 모두가 실제로 유효한 작업부하를 수행하는 기본구성요소이다. 하나가 고장나면 그것의 작업부하는 다른 구성요소들에 재분배된다. 이 방법은 정상적으로 휴식하는 구성요소들이 없기때문에 비용이 효률적이다. 그러나 이어받기는 더 많은 시간을 소비한다.
- **고장허용** 이것은 이미 실례 9.4의 NVP에서 언급되였다. 이것은 N개의 구성요소들이 N배 더 많은 가격으로 한 구성요소의 성능만을 주기때문에 가장 비싼 형태이다. 그러나 N-1개 구성요소들의 고장은 감춰진다(사용자에게 보이지 않는다.).
- **failover** 이것은 상업응용들을 위한 현재의 클라스터들에서 요구되는 가장 중요한 특징이다. 하나의 구성요소가 고장날 때 이 기술은 나머지체계가 원래 고장난 구성요소가 제공하던 봉사를 이어 받도록 한다. failover기구는 고장진단, 고장통지, 고장회복과 같은 여러 기능들을 제공해야 한다.

고장진단은 고장의 발견과 고장을 일으킨 고장난 구성요소의 위치에 귀착된다. 대중적으로 사용되는 기술은 심장박자(heartbeat)인데 클라스터마디들은 심장박자통보문의 흐름을 다른 마디로 내보낸다. 만일 체계가 마디로부터 심장박자통보문의 흐름을 받지 못하면 그것은 마디나 망의 접속이 고장났다고 결론할수 있다.

실례 9.6. 2개 망클라스터에서 고장진단과 회복

클라스터는 클라스터의 마디들을 접속하는 2개의 망을 가진다.

하나의 마디가 주마디로 지적된다. 매 마디는 두 망을 통하여 주마디에 주기적으로(매 10s마다) 심장박자통보문을 보내는 심장박자데몬을 가진다. 주마디는 한박자(10s)동안 마디로부터 통보문들을 받지 못하면 고장을 발견하고 다음의 진단을 내린다.

- 만일 주마디가 한마디로부터 하나의 망을 통하여 심장박자를 받고 다른 마디로부터 받지 못하면 두 망중의 한 망에 대한 그 마디의 접속이 고장났다.
- 만일 주마디가 두망을 통하여 심장박자를 받지 못한다면 그 마디가 고장났다. 두 망이 동시에 고장나는 경우는 무시해도 된다고 본다.

이 실례에서 고장진단은 간단하지만 여러개의 함정이 있다.

주마디가 고장나면 어떻게 되겠는가?

10s라는 심장박자는 너무 긴가 아니면 너무 짧은가?

심장박자통보문이 망에서 잃어 지면(실례로 망혼잡으로 인하여) 어떻게 되겠는가?

이 방법이 수백개의 마디들을 수용할수 있겠는가?

실제적인 높은 유용체계들은 이 문제들을 처리해야 한다.

대중적인 방법은 부하정보를 나르는 심장박자통보문을 사용하여 주마디가 마디로부터 심장박자를 받으면 주마디는 그 마디가 살아 있다는것뿐아니라 그 마디의 자원리용상태까지도 알게 하는것이다. 그러한 부하정보는 부하평형이나 일감관리에도 유용하다.

일단 고장이 진단되면 체계는 구성요소들에 고장사건을 알도록 통지한다.

고장통지는 주마디가 이 정보를 가져야 할 유일한 마디가 아니기때문에 필요하다. 자원관리자는 작업부하를 재배정하고 그 마디의 남은 작업부하를 이어 받도록 요구한다. 체계관리자는 그 마디를 수리하는데 적당한 동작들을 초기화할수 있도록 경보를 보낼것을 요구한다.

실례로 한 마디가 고장난 경우에 령역이름봉사기(DNS)는 자기가 더 많은 사용자들을 그 마디에 접속하지 않도록 알려 줄것을 요구한다.

회복방법

고장회복은 고장난 구성요소의 작업부하를 이어 받는데 필요한 동작들에 귀착된다.

2가지 형태의 회복기술이 있다.

뒤방향회복에서는 클라스터에서 실행하는 프로쎄스들이 하나의 모순이 없는 상태(검사점이라고 한다.)를 안전한 기억에 주기적으로 보존한다. 고장후 체계는 고장난 구성요소를 분리시키기 위해 재구성되며 이전의 검사점을 복귀시키고 정상동작을 다시 시작한

다. 이것을 **복구**라고 부른다.

뒤방향회복은 상대적으로 응용독립의 이식가능한 형태로 실현하기 쉬우며 널리 사용되였다. 그러나 복구는 쓸데 없는 실행을 포함한다. 복구시간을 허용할수 없는 실시간체계에서와 같이 실행시간이 극히 중대하면 앞방향회복방법이 사용되여야 한다.

이 방법에서는 고장날 때 체계가 이전의 검사점으로 복구되지 않는다. 대신에 체계는 타당한 체계상태를 재구성하기 위한 고장진단정보를 리용하여 실행을 계속한다. 앞방향회복은 응용의존적이며 여분의 하드웨어를 요구한다.

실례 9.7. MTTF와 MTTR, 고장비용의 해석

유용성지원이 거의 없는 클라스터를 생각하자. 마디가 고장날 때 다음과 같은 사건들이 순차로 일어 난다.

(1) 전체 체계가 닫겨 지며 전원이 차단된다.
(2) 고장마디가 하드웨어마디이면 고장마디는 교체된다.
(3) 체계에 전원이 공급되고 다시 초기적재된다.
(4) 사용자의 응용이 재적재되고 처음부터 다시 실행된다.

클라스터의 마디들중 하나가 100시간마다 실패한다고 하자. 클라스터의 다른 부분들은 절대로 고장나지 않는다. 걸음 1~3은 다해서 2시간 걸린다. 걸음 4에 대한 평균시간은 2시간이다. 클라스터의 유용성은 얼마인가? 매 1시간의 정지시간이 82500$(실례 9.1)를 잃게 한다면 년간 고장비용은 얼마인가?

답:

클라스터의 MTTF는 100시간이고 MTTR는 2+2=4시간이다.

식 (9.1)로부터 유용성은 100/104=96.15%이다. 이것은 한해에 337시간의 정지시간에 대응되며 고장비용은 82500×337$ 즉 27M$이상이다.

실례 9.8. 콤퓨터들의 클라스터에 대한 유용성 및 비용해석

실례 9.7을 반복하는데 클라스터가 많이 증가된 유용성지원을 가진다고 하자. 마디가 고장나면 그것의 작업부하는 자동적으로 다른 마디들로 이어 진다. 고장여유시간은 6분뿐이다. 그동안 클라스터는 작업중교체(hot swap)능력을 가진다. 즉 고장마디는 클라스터에서 떨어 져 수리되며 다시 련결되고 다시 초기적재되여 클라스터에 다시 결합된다. 이리상적인 클라스터의 유용성은 얼마이며 년간 고장비용은 얼마인가?

답:

클라스터의 MTTF는 여전히 100시간이지만 MTTR는 클라스터가 고장난 마디가 수리되고 있는 동안에도 리용가능하므로 0.1시간으로 낮아 진다.

식 (9.1)로부터 유용성은 100/100.5=99.9%이다. 이것은 해마다 8.75시간의 정지시간에 대응하며 고장비용은 82500×8.75=722000$로서 실례 9.7의 설계에서의 고장비용보다

27M/772K=38배 작다.

9. 2. 3. 검사점설정과 고장회복

클라스터체계의 유용성을 높이기 위하여 개발해야 할 2가지 기술이 있다.

검사점설정의 기본개념으로부터 시작하자.

기본개념

검사점설정은 실행하는 프로그람의 상태를 안전한 기억에 주기적으로 보존하는 프로쎄스(process)이다. 이것을 리용하여 체계는 고장이 일어 난후 회복할수 있다.

보존된 매개 프로그람상태를 **검사점**이라고 한다.

보존된 상태를 포함하는 디스크파일을 **검사점파일**이라고 부른다.

현재의 검사점설정쏘프트웨어는 프로그람의 상태를 디스크에 모두 보존하지만 성능을 높이기 위하여 안정기억(stable storage)대신에 마디의 기억기들을 사용하기 위한 연구가 진행중에 있다[33]. 검사점설정기술들은 유용성뿐아니라 프로그람의 오유수정과 프로쎄스의 이동, 부하평형에도 쓸모가 있다.

많은 일감관리체계들과 일부 조작체계들은 일정한 정도로 검사점설정을 지원한다. Wed자원은 Condor와 libckpt와 같은 일부 공개령역 쏘프트웨어를 포함하는 다수의 검사점관련Web싸이트들에 대한 지시자들을 포함한다.

이 부분에서는 검사점쏘프트웨어의 설계자와 사용자에게 중요한 문제들을 론의한다.

핵심부와 서고, 응용의 수준들

검사점은 조작체계에 의하여 핵심부수준에서 실현될수 있다. 여기서 OS는 투명하게 검사점을 찾으며 프로쎄스들을 다시 시작한다. 이것은 사용자들에게 있어서 리상적이다. 그러나 검사점설정은 대부분의 조작체계들 특히 병렬프로그람들에서 지원되지 않는다.

보다 덜 투명한 방법은 사용자코드를 사용자공간에서 검사점설정서고와 련결한다. 검사점설정과 사용자응용들은 수정되지 않아도 된다는 우점을 가지므로 널리 사용된다. 기본문제는 대부분의 현존 검사점설정서고들이 정적이라는것 즉 응용의 원천코드(또는 적어도 목적코드)가 리용가능해야 한다는것이다. 그것은 응용이 실행가능한 코드의 형식이면 동작하지 않는다.

세번째 방법은 사용자(또는 콤파일러)가 검사점기능들을 응용에 삽입할것을 요구한다. 따라서 응용은 수정되여야 하며 투명성이 상실된다. 그러나 그것은 사용자가 검사점을 만드는 위치를 특정화할수 있다는 우점을 가진다. 이것은 검사점설정의 부가처리를 줄이는데 쓸모 있다.

검사점설정은 시간과 기억의 부가처리를 초래한다. 이 부가처리들은 아래에서 정의된다.

검사점부가처리

프로그람이 실행되는 동안 그것의 상태는 여러번 보존된다. 한개의 검사점을 보존하는데 소비된 시간을 t_c 로 표시하자. 기억기의 부가처리는 검사점설정에 필요한 여분의

기억기 및 디스크공간이다. 시간 및 기억의 부가처리는 검사점파일의 크기에 의존한다. 이 부가처리들은 특히 큰 기억기공간을 요구하는 응용들에서 실제로 존재할수 있다. 이 부가처리를 줄이기 위한 여러가지 기술들이 제안되였다.

최적검사점간격의 선택

2개의 검사점사이의 시간구간을 **검사점간격**이라고 한다. 이 간격을 더 크게 함으로써 검사점의 시간부가처리를 줄일수 있다. 그러나 이것은 고장후에 더 긴 재계산시간을 포함한다.

Wong과 Franklin[648]은 리론적인 모형에 기초하여 최적검사점간격을 주는 다음과 같은 식을 유도하였다.

$$\text{최적검사점간격} = \sqrt{(MTTF \cdot t_c)/h}$$

여기서 $MTTF$는 체계가 고장나는 평균시간이고 t_c는 하나의 검사점을 보존하는데 소비된 시간이며 h는 체계가 고장나기전에 한개의 검사점간격사이에 수행한 표준계산의 평균률이다.

파라메터 h는 항상 $0 \le h \le$의 범위에 있다.

체계가 원상회복된후에 재계산에 $h \times$(검사점간격)시간이 걸린다. 이 시간파라메터들은 그림 9-6에서 설명된다.

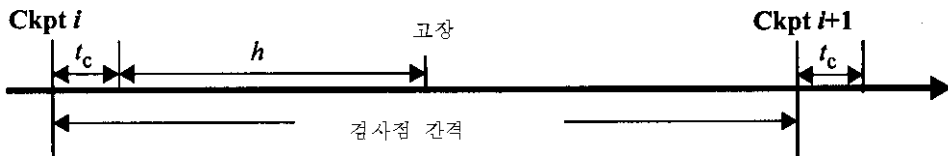



그림 9-6. 2개의 검사점사이의 시간파라메터들

증분검사점

증분검사점방법[503, 504, 505]은 매개 검사점에 충분한 상태를 보존하지 않고 이전의 검사점으로부터 변화된 상태부분만을 보존한다. 그러나 이전의 검사점파일들을 관심해야 한다.

완전상태검사점설정에서는 디스크에 하나의 검사점파일만이 보존될것을 요구한다. 다음의 검사점들은 단순히 이 파일에 덧씌여 진다.

증분검사점설정에서는 완전한 상태가 여러개의 파일들에 퍼져 있으므로 이전의 파일들이 보존될것을 요구한다. 그러므로 전체적인 기억요구가 더 크다.

포크식검사점설정

대부분의 검사점설정방법들은 검사점설정이 진행되는 동안 일반계산이 차단(정지)되는 차단식이다. 충분한 기억기공간이 있으면 검사점부가처리는 기억기에 프로그람의 상태를 복사하고 일반계산과 함께 검사점설정을 동시에 수행하는 또 다른 비동기스레드를

포함함으로써 감소될수 있다.

계산과 검사점설정을 겹치게 하는 간단한 방법은 Unix의 fork()체계호출을 사용하는 것이다.

포크식자손프로쎄스는 부모프로쎄스의 주소공간을 복사하고 거기서 검사점으로 만든다. 한편 부모프로쎄스는 실행을 계속한다. 겹치기는 검사점설정이 디스크 I/0집약적이기때문에 달성될수 있다. 그이상의 최량화는 copy−on−write기구[415]를 사용하는것이다.

검사점의 압축

또 다른 착상은 표준적인 압축알고리듬들을 리용하여 매 검사점을 압축하는것이다. 그러나 압축은 압축의 부가처리가 작고 검사점이 크게 줄어 들수 있을때만 진행한다. 압축은 단일처리기에는 비효률적이고 디스크경쟁을 가지는 병렬체계들에서 효률적이라는것을 보여 주는 일부 증거들이 있다.[505]

사용자지시검사점설정

검사점의 부가처리는 때때로 사용자가 언제 보존하고 무엇을 보존하며 무엇을 보존하지 않는가 하는것을 체계에 알리는 코드들(실례로 서고나 체계호출)을 삽입하면 실제적으로 감소될수 있다. 사용자지시검사화에 대한 개괄은 [503]을 참고할것 .

무엇을 검사점으로

검사점의 정확한 내용은 무엇이겠는가?

그것은 바로 체계를 회복하는데 충분한 정보를 포함한다. 2.2에서 론의한것과 같이 프로쎄스의 상태에는 그것의 자료상태와 조종상태가 포함된다.

Unix의 프로쎄스에 대하여 이 상태들은 본문(코드)과 자료, 탄창토막들, 프로쎄스의 서술자를 포함하는 주소공간에 기억된다. 충분한 상태를 보존하고 되살리는것은 비용이 많이 들며 때때로 불가능하다. 그것들은 또한 종종 불필요하다. 실례로 많은 응용들에서 프로쎄스의 ID와 부모프로쎄스의 ID는 회복가능하지 않으며 대부분의 검사점설정체계들의 부분적인 상태를 보존한다. 실례로 코드토막은 대부분의 응용들에서 변하지 않기때문에 보통 보존되지 않는다.

일반성

어떤 종류의 응용들에 대하여 검사점을 만들수 있는가?

현존 검사점설정방법들은 프로그람들이 잘 동작할것을 요구한다. 그 정확한 의미는 서로 다른 방법들에서 서로 다르다.

최소한 잘 트랜잭션된 프로그람은 프로쎄스 ID의 수값과 같은 회복할수 없는 상태정보의 정확한 내용을 요구하지 말아야 한다.

Condor묶음[608]은 자손프로쎄스들을 창조하지 않고 신호와 관흐름, sockets, 파일들과 같은 Unix의 마디호상간의 통신방법들을 사용하는 다른 프로쎄스들과 통신하지 않는 그러한 프로그람들에 대해서만 검사점을 만들수 있다.

Libokpt의 검사점서고[503]는 열린 파일들의 표만을 보존하고 다른 체계상태정보는

보존하지 않는다.

임의의 검사점설정쏘프트웨어를 사용할 때 중요한것은 어떤 프로그람들이 잘 동작하는가 하는것이다. 왜냐하면 서틀게 동작하는 프로그람은 정확히 검사점이 만들어 지지 않기때문이다.

편리성

검사점설정쏘프트웨어를 사용하는것이 얼마나 쉬운가?

리상적으로 검사점설정기능은 조작체계에 통합되여야 사용자의 응용들이 자동적으로 투명하게 보존되고 복구될수 있다. 그러나 대부분의 현존 조작체계들은 검사점설정을 잘 지원하지 않는다.

다음으로 가장 좋은 방법은 사용자가 실행하는 응용과 련결할수 있는 실시간서고를 지원하는것이다. 그러나 서고가 동적이든 정적이든 사용자에 따라 큰 차이를 가질수 있다.

Condor의 이전 판본들에서 사용자들은 정적검사점서고와 목적코드를 련결해야 한다. 이것은 응용이 3부류의 실행가능코드이면 불가능하다.

Condor의 판본 6에서는 Sun Solaris에서 이 제한을 없애기 위하여 동적련결서고가 제공된다.

우의 방법들은 사용자가 자기의 코드로 수정하지 않아도 되기때문에 투명한 검사점설정이라고 부른다.

다른 방법들은 투명하지 않다.

가장 간단한것은 Libckpt서고에서 설명되는데 이것은 사용자가 원천코드의 한행을 변화시킬것을 즉 main()함수를 ckpt-target()로 바꿀것을 요구한다. 투명하지 않은 다른 방법들은 사용자들이 원천코드에 검사점지령들을 삽입할것을 요구한다. 이것은 사용자에게는 불편하지만 이 불편성은 사용자지시점화의 진보에 반비례하여 증대된다.

검사점설정병렬프로그람

이제는 검사점설정병렬프로그람들을 보자.

병렬프로그람의 상태는 보통 순차프로그람의 상태보다 더 크다. 왜냐하면 그것이 개별적인 프로세스들의 상태의 모임과 통신망의 상태로 구성되기때문이다. 병행은 또한 여러가지 시간 및 일치성문제들을 끌어 들인다.

실례 9.9. 병렬프로그람의 검사점설정

3개 프로쎄스병렬프로그람의 검사점설정을 그림 9-7에서 설명한다.

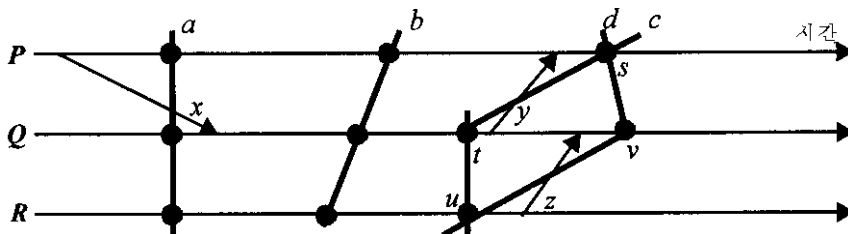



그림 9-7. 병렬프로그람의 일치성 및 불일치성검사점들

화살 *x*, *y*, *z*는 프로쎄스들사이의 점대점통신을 표시한다. 3개의 굵은 선 *a*, *b*, *c*는 3개의 대역 snapshot(또는 간단히 snapshot)를 표시한다. 여기서 하나의 대역 snapshot는 매 프로쎄스로부터 1개씩 취한 검사점(점으로 표시됨)들의 모임이다.

그밖에 일부 통신상태가 보존되여야 한다. snapshot시간선사이의 부분들은 프로쎄스가 어디에 (국부)검사점을 가져야 하는가를 가리킨다. 그러므로 프로그람의 snapshot *c*는 3개의 국부검사점들 즉 프로쎄스 *P*, *Q*, *R*에 대하여 각각 *s*, *t*, *u*와 통신 *y*의 복사로 이루어 진다.

일치성 snapshot

다른 프로쎄스의 검사점은 계속 보내고 한 프로쎄스의 검사점은 받아 들이는 통보문이 없을 때 대역 snapshot는 일치한다고 한다. 그라프적으로 이것은 화살이 오른쪽에서 왼쪽으로 snapshot선과 교차하지 않는다는것과 동등한다.

이 정의에 의하여 snapshot a는 화살 x가 왼쪽에서 오른쪽으로 향하기때문에 일치한다. 그러나 snapshot c는 y가 오른쪽으로부터 왼쪽으로 가므로 불일치이다.

Netzer와 Xu[461]는 모든 검사점들이 주어 졌을 때 2개의 검사점이 하나의 일치성 snapshot에 속하기 위한 필요충분조건을 증명하였다. 조건은 이 2개의 검사점사이에 어떤 Z자모양의 경로도 없어야 한다는것이다. 실례로 검사점 u와 s는 하나의 일치성대역 snapshot에 속할수 없다.

일치성 snapshot에 대한 더 강한 정의는 화살들이 snapshot를 교차하지 않을것을 요구한다. 이 정의에 의하여 그림 9-7에서는 snapshot b만이 일치한다.

동등검사점 대 독립검사점

병렬프로그람에 대한 검사점설정방법들은 2가지 형태로 분류할수 있다.

동등검사점설정(일치성검사점설정이라고도 한다.)에서 병렬프로그람은 동결되고 모든 프로쎄스들은 동시에 검사점이 만들어 진다. 한가지 실례는 CoCheck[608]이다.

독립검사점설정에서 프로쎄스들은 호상 독립적으로 검사점이 만들어 진다.

이 두가지 형태들은 여러가지 방법으로 결합될수 있다.

동등검사점설정은 실현하기 힘들며 큰 부가처리를 일으키는 경향이 있다.

독립검사점설정은 작은 부가처리를 가지며 순차프로그람들에 대한 현존 검사점설정방법들을 리용할수 있다. 그러나 그것은 도미노효과문제를 풀어야 한다.

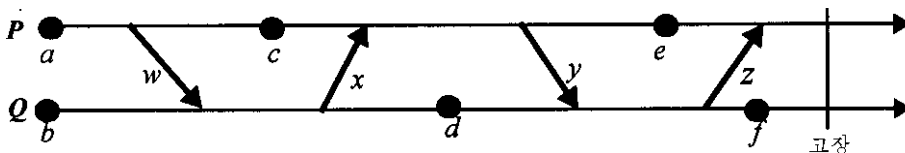



그림 9-8. 검사점들의 렬에서 도미노효과의 설명

도미노효과

그림 9-8을 고찰하자.

체계가 고장나서 프로쎄스 *P*를 그것의 국부검사점 *e*로 복구한다고 하자. 이것은 통신 z에 의하여 프로쎄스 *Q*로부터 다시 통보문을 보내올것을 요구한다. 이를 위해 *Q*는 검사점 d로 복구되여야 한다. 이때 *Q*는 통신 *y*에 의해 프로쎄스 *P*로부터 보내오는 통보문을 요구한다. 그리하여 *P*는 *c*에로 복구될것을 요구한다. 결국 *P*와 *Q*는 다 초기상태 *a*와 *b*로 복구될것을 요구하며 검사점설정을 쓸모없게 만든다.

도미노효과를 피하기 위하여 독립검사점설정은 통보문기록하기와 함께 확대된다. 착상은 검사점을 만드는 동안 개별적인 프로쎄스들이 자기의 국부검사점들을 독립적으로 보존할뿐아니라 통보문들을 기록(log)에 보존시키는것이다.

그림 9-8에서 대역 snapshot는 고장직전에 국부검사점들 *e*와 *f*, 그리고 통신 *z*에 대한 통보문기록으로 구성된다. 회복될 때 프로쎄스들은 자기의 검사점들 *e*와 *f*로 복구되며 통신 *z*가 다시 진행된다.

9. 3. 단일체계영상에 대한 지원

단일체계영상은 SMP나 워크스테이션에서와 같이 기억기에 상주하고 있는 조작체계형태의 단일한 복사를 의미하지 않는다. 오히려 그것은 아래에서 특징 짓는것과 같은 단일체계의 환상을 의미한다.

- **단일체계** 여러 처리기들을 가지는 전체 클라스터는 사용자들에게 하나의 체계로 보인다. 사용자는 《5개의 처리기를 리용하여 나의 응용을 실행하라.》고 말할수 있다. 이것은 분산체계와 다르다.
- **단일조종** 론리적으로 말단사용자나 체계사용자는 단일한 대면부를 가지고 한 위치에서 봉사를 리용한다. 실례로 사용자는 묶음일감들을 대기렬들중 하나로 제출하며 체계관리자는 클라스터의 모든 하드웨어 및 쏘프트웨어구성요소들을 구성한다.
- **균형성** 사용자는 임의의 마디로부터 클라스터의 봉사를 받아 사용할수 있다. 다시 말하여 클라스터의 모든 봉사들과 기능들이 사용접근규칙에 의해 보호되는 것들을 제외하고 모든 마디와 모든 사용자들에게 균형적이다.
- **위치투명성** 사용자는 최종적으로 봉사를 제공하는 물리장치들이 있는 곳을 모른다. 실례로 사용자는 임의의 클라스터의 마디에 련결된 테프장치를 마치도 물리적으로 국부마디에 련결되여 있는것처럼 사용할수 있다. 그러나 일부 성능상 차이는 있을수 있다.

9. 3. 1. 단일체계영상의 층

SSI를 가지게 되는 기본적인 동기는 그것이 잘 알려 진 워크스테이션과 같이 사용되고 조종되며 관리된다는것이다. 단일체계영상에서 단어 단일은 때때로 대역 또는 집중과 같은 뜻으로 불리운다. 실례로 대역파일체계는 사용자가 임의의 마디로부터 접근할수 있는 단일한 파일계층을 의미한다.

단일조종점은 한 운영자가 클라스터체계를 감시하고 구성하게 한다.

단일체계의 환상은 있지만 클라스터의 봉사나 기능은 흔히 분산적으로 여러 구성요소들의 협조를 통하여 실현된다.

SSI기술의 기본요구(우월성)는 그것이 분산실현의 성능상 우점과 단일형태의 유용성의 우점들을 다 제공한다는것이다.

정의 9.2. 프로쎄스 *P*의 관점에서 클라스터의 마디들은 3가지 형태로 분류될수 있다. 프로쎄스 *P*의 원천(home)마디는 프로쎄스 *P*가 창조될 때 상주하는 마디이다. 프로쎄스 *P*의 국부마디는 *P*가 현재 상주하고 있는 마디이다. 다른 모든 마디들은 *P*에 원격마디로 된다.

정의 9.3. 클라스터의 마디들은 서로 다른 요구에 맞게 구성될수 있다. 호스트마디는 telnet와 원격접속(또는 ftp와 http)을 통하여 사용자접속(login)을 봉사한다. 콤퓨터마디는 계산일감들을 수행하는 마디이다. I/0마디는 파일 I/0요구들에 봉사하는 마디이다. 클라스터가 큰 공유디스크나 테프단들을 가지면 그것들은 보통 I/0마디에 물리적으로 련결된다.

매 프로쎄스에 대하여 하나의 원천마디가 있으며 이것은 프로쎄스가 살아 있는 전기간에 걸쳐 고정된다. 임의의 시각에 호스트마디이거나 아닌 오직 하나의 국부마디만이 있다. 프로쎄스의 국부마디와 원격마디들은 프로쎄스가 이동하면 변한다. 마디는 여러 기능을 제공하도록 구성될수 있다.

실례로 마디는 동시에 호스트와 I/0마디, 콤퓨터마디로 지명될수 있다.

문헌은 혼합된 이름을 사용한다. 실례로 호스트는 때때로 임의의 마디형태를 참조하는데 사용된다. 즉 호스트는 마디와 같다. 호스트 또는 I/0마디는 때때로 그것이 어떤 봉사들을 제공한다는것을 가리키기 위하여 봉사기마디라고 부른다.

SSI의 환상은 여러 층으로 얻어 지는데 그것들중 4개를 아래에서 론의한다. 이 층들은 서로 겹칠수 있다.

- **응용쏘프트웨어층** 2개의 실례는 Web browser와 여러가지 병렬자료기지이다. 사용자는 응용을 통하여 단일체계영상을 보며 클라스터를 사용하고 있다는것은 알지조차 못한다. 이 방법은 클라스터를 위하여 워크스테이션이나 SMP의 응용들에 대한 수정을 요구한다.
- **하드웨어 또는 핵심부층** 리상적으로 SSI는 조작체계나 하드웨어에 의해 제공된다. 아쉽게도 이것은 아직까지 실현되지 못했다. 더우기 이질클라스터들에 파일체계형태를 제공하는것은 극히 곤난하다. 대부분의 하드웨어구조들과 조작체계들을 전용화하여서만 이 방법을 사용할수 있다.
- **핵심부웃층** 가장 생활력 있는 방법은 그림 9-2와 같이 SSI층을 핵심부의 바로 우에 구성하는것이다. 이 방법은 가동환경독립이고 응용의 수정을 요구하지 않기때문에 전망이 있다. 많은 클라스트의 일감관리체계들은 이미 이 방법을 받아

들이였다. 이에 대하여서는 9.5에서 론의한다.

9. 3. 2. 단일입력자료와 단일파일계층

단일체계영상은 대단히 풍부한 개념으로서 단일입력자료, 단일파일계층, 단일I/0공간, 단일망화, 단일조종점, 단일일감관리체계, 단일기억기공간, 단일프로쎄스공간 등으로 이루어 진다.

단일입력자료

단일입력자료는 사용자들이 클라스터가 접속대화(login session)를 지원하는 여러개의 물리적호스트를 가진다고 해도 하나의 가상호스트로서 클라스터에(telnet나 rlogin, http를 통하여) 접속할수 있게 한다.

체계는 부하평형을 위하여 사용자의 접속 및 접속요청들을 서로 다른 물리적호스트들에 투명하게 분산시킨다.

왜 수천개의 사용자대화(session)를 봉사하는 클라스터가 필요한가고 이상하게 여길수 있다. 그 대답은 클라스터들이 대형콤퓨터들과 초고속콤퓨터들을 대신할수 있다는것이다. 또한 인터네트클라스터봉사기에서는 수천개의 http 또는 ftp요청들이 동시에 들어 올수도 있다.

여러 호스트들을 가지는 단일입력자료를 구성하는것은 사소한 문제가 아니다. 많은 문제들이 해결되여야 한다.

다음의것들이 바로 그 부분적인 목록이다.

- **원천등록부** 사용자의 원천등록부를 어디에 넣겠는가?
- **인증** 사용자의 접속을 어떻게 인증하겠는가?
- **다중접속** 같은 사용자가 같은 사용자회계에로의 여러개의 대화를 열면 어떻게 되는가?
- **호스트고장** 하나 또는 그이상의 호스트고장을 어떻게 처리하겠는가?

단일입력자료를 실현하는 간단한 방법을 그림 9-9에서 설명한다.

클라스터의 4개 마디들이 사용자의 접속요청을 받아 들이는 호스트마디로 사용된다. 하나의 사용자만 보여 주고 있지만 수천의 사용자들이 같은 형태로 클라스터에 접속할수 있다.

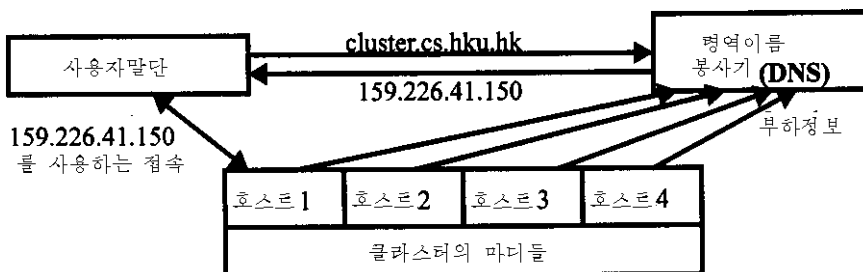



그림 9-9. 부하평형식령역이름봉사기를 사용하는 단일입력자료의 출현

사용자가 클라스터에 접속하려면 클라스터체계의 기호적이름을 사용하여 《telnet.cluster.cs.hku.hk》와 같은 표준적인 Unix지령을 내보내야 한다.

기호적이름은 DNS에 의해 변환되여 부하가 가장 적은 마디의 IP주소 159.226.150와 함께 귀환된다. 이것은 마디 Hostl에서 수행된다.

그다음 사용자는 이 IP주소를 사용하여 접속한다. DNS는 부하평형트랜잭션결정을 내리기 위하여 호스트마디들로부터 주기적으로 부하정보를 받는다.

리상적인 경우 200명의 사용자가 동시에 접속하면 접속대화들이 호스트당 50사용자씩 4개의 호스트들에 평등하게 분배된다.

실례 9.11. 원천등록부문제의 해답

원천등록부문제에 대한 한가지 해답은 모든 사용자들의 원천등록부들을 클라스터의 안정기억에서 유지하는것이다(안정기억의 개념은 단일파일계층에 대하여 이야기할 때 더 론의한다.).

문제는 원천등록부에로의 접근이 원격동작이여서 비효률적이라는것이다.

방도는 모든 호스트에서 매개 사용자의 원천등록부의 복사를 유지하는것이다. 그러나 이 해답은 2개의 주요부족점을 가진다.

- 디스크요구가 너무 커질수 있다. 클라스터가 1000개의 사용자계정을 지원해야 하고 매 사용자는 20MB의 디스크몫을 가진다고 하자. 그러면 이제는 매 호스트의 국부디스크가 적어도 20GB의 용량을 가져야 한다는것을 의미하는데 이것은 너무 크다.
- 일치성을 유지하자면 사용자가 접속을 끊을 때 가장 최근의 원천등록부는 모든 호스트들에 방송되여야 한다.

단일파일계층

이 책에서는 단일파일계층(single file hierarchy)이라는 술어를 국부 및 대역디스크들과 다른 파일장치들(실례로 테프)을 투명하게 통합한 단일하고 거대한 파일체계형태를 의미하는것으로 리해한다. 다시 말해서 사용자가 요구하는 모든 파일들은 뿌리등록부의 어떤 부분등록부들에 기억되며 open, read 등과 같은 보통의 Unix호출을 통하여 접근될수 있다는것이다. 여러 파일체계들이 뿌리등록부의 부분등록부들과 워크스테이션에 존재할수 있다는 사실과 혼돈하지 말아야 한다.

단일파일계층의 기능들은 망파일체계(NFS)[527]와 안드레이파일체계(AFS)[453]와 같은 현존 분사파일체계들에 의하여 이미 부분적으로 제공되였다.

임의의 프로쎄스의 관점에서 파일들은 그림 19.10에서와 같이 클라스터의 3가지 형태의 위치에 상주할수 있다. 국부기억은 프로쎄스의 국부마디의 디스크이다.

원격마디의 디스크들은 원격기억이다. 안정기억은 다음과 같이 특징 지어 진다.

- 그것은 영구적이다. 이것은 일단 안정기억에 씌여 진 자료가 지어 클라스터를 닫은 후에도 적어도 일정한 시간동안(실례로 한주동안) 거기에 있어야 한다는것

을 의미한다.

- 그것은 여유와 테프에로의 주기적인 복제를 사용하여 어느 정도로 고장을 허용한다.

론리적으로 집중식의 안정기억을 그림 9-10에서 보여 준다. 안정기억의 파일들은 대역파일, 국부기억의 파일들은 국부파일, 원격기억의 파일들은 **원격파일**이라고 부른다. 안정기억은 하나의 집중식의 큰 RAID디스크로 실현될수 있다. 그러나 그것은 클라스터의 마디들의 국부디스크들을 사용하여 분산적으로 실현될수도 있다.

첫번째 방법은 하나의 큰 디스크를 사용하는데 이것은 단일고장점이며 가능한 성능상의 목적으로 된다. 뒤의 방법은 실현하기는 더 힘들지만 유력하게 더 경제적이고 더 효률적이며 유용하다.

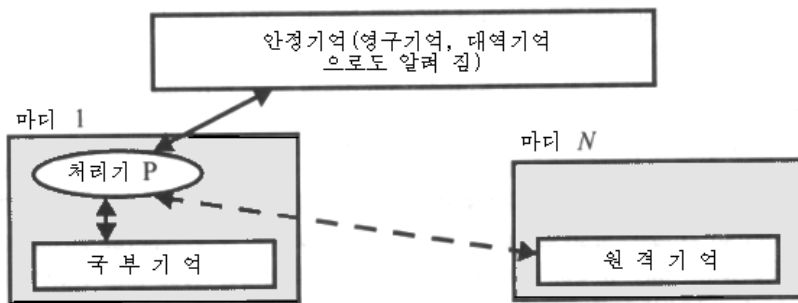



그림 9-10. 단일파일계층에서 기억의 3가지 형태

실례 9.12. Unix클라스터의 단일파일계층

많은 클라스터와 MPP체계들에서 체계가 단일파일계층의 다음의 등록부들을 사용자의 프로쎄스들에 보이게 하는것이 전통적이다.

- /usr와 /usr/locl 등과 같은 전통적인 Unix워크스테이션에서와 같은 보통의 체계등록부들.
- 작은 디스크몫(1-20MB)을 가지는 사용자의 원천등록부. 사용자는 자기의 코드파일들과 다른 파일들을 기억한다. 그러나 큰 자료파일은 다른곳에 기억시켜야 한다.
- 모든 사용자와 모든 프로쎄스들에 공유되는 대역등록부/scratch(또는 /global scratch). 이 등록부는 여러 기가바이트의 큰 디스크공간을 가진다. 사용은 자기의 큰 파일들을 여기에 기억할수 있다.
- 클라스터체계에서 프로쎄스는 흔히 local scratch로 표시하는 국부디스크의 특별한 등록부에 접근할수 있다. 이 등록부는 중간용량(실례로 수백메가바이트)이며 global scratch등록부보다 더 빨리 접근할수 있다.

파일들에 대한 투명성

여기서 술어 투명성은 파일에 접근하기 위한 fopen, fread, fwrite와 같은 전통적인

Unix체계 및 서고호출들을 사용할수 있다는것을 의미한다.

하나의 클라스터에 여러개의 국부scratch등록부들이 있다는데 주의해야 한다. 원격마디의 국부scratch등록부들은 단일파일계층에 없으며 프로쎄스에 직접 보이지 않는다.

사용자의 프로쎄스는 마디의 이름과 파일이름을 다 명기함으로써 여전히 rcp와 같은 지령들 또는 일부 특수한 서고함수들에 의해 그것들에 접근할수 있다.

이름 scratch는 그 기억이 림시정보기억에 대해서는 지워쓰기로서 동작한다는것을 가리킨다. 국부scratch의 정보는 일단 사용자가 접속을 해제하면 잃어 진다. 대역scratch의 파일들은 사용자가 접속을 해제한후에도 계속 유지되지만 사전에 결정된 시간동안 접근되지 않으면 체계에 의해 지워 진다. 이것은 디스크공간을 다른 사용자를 위하여 해방하는것이다. 시간길이는 체계관리자가 설정하는데 보통 1일~여러주의 범위에 있을수 있다. 일부 체계들은 대역scratch를 주기적으로 또는 임의의 파일을 지우기전에 테프에 복제한다.

요구되는 SSI의 속성들

단일파일계층은 SSI속성들을 가져야 한다. 이것은 파일체계들에 대하여 다음과 같이 되풀이된다.

- **단일체계** 사용자의 관점에서 바로 하나의 파일계층만이 있다.
- **균형성** 사용자는 클라스터의 봉사를 리용하여 임의의 마디로부터 대역기억(실례로 /scratch)에 접근할수 있다. 다시 말하여 모든 파일봉사들과 기능들은 접근규칙에 의해 보호되는것들을 제외하고 모든 마디들과 모든 사용자들에게 균형적이다.
- **위치투명성** 사용자는 최종적으로 봉사를 제공하는 물리장치가 있는곳을 모른다. 실례로 사용자는 임의의 클라스터마디에 련결된 RAID를 마치도 그것이 물리적으로 국부마디에 련결된것처럼 사용할수 있다. 그러나 일부 성능상의 차이는 있을수 있다.

하나의 파일연산은 ACID속성들(정의 1.3)을 만족하는 하나의 트랜잭션이다. 클라스터파일체계는 Unix의 의미들을 유지한다. 즉 모든 파일연산(fopen, fread, fclose등)은 하나의 트랜잭션이다. fwrite가 파일을 수정한후에 fread가 같은 파일에 접근할 때 fread는 갱신된 값을 얻게 된다. 그러나 현재의 분산파일체계들은 Unix의 의미를 완전히 따르지 않는다. 그것들중 일부는 close 또는 flush때에만 파일을 갱신한다.

클라스터의 대역기억을 조직하기 위한 여러 방법들이 제기되였다.

하나의 극단은 하나의 큰 RAID를 호스트하는 단일한 파일봉사기를 사용하는것이다. 이 방법은 간단해서 현존 쏘프트웨어(실례로 NFS)로 쉽게 실현될수 있다. 그러나 파일봉사는 성능상의 병목 및 단일고장점으로 된다.

다른 극단은 대역기억을 형성하는데 모든 마디들의 국부디스크를 리용하는것이다. 이것은 단일파일봉사기의 성능 및 유용성문제들을 해결할수 있다. 그러나 오늘 이 목적에 쉽게 리용할수 있는 성숙된 쏘프트웨어는 없다.

단일파일계층의 실현

대부분의 파일체계들에서 캐쉬는 성능을 개선하는데 널리 사용된다. 대중적인 방법

은 의뢰기측 캐쉬이다.

파일연산을 내보내는 마디는 자주 사용되는 파일토막들을 캐쉬에 요구한다. 그러나 캐쉬는 일관성(또는 일치성)문제가 생기게 한다. DSM를 위하여 개발한 캐쉬일관성기술들이 여기에 사용될수 있다. 단일파일계층형태를 실현하는 여러가지 방법이 있다. 3가지 방법들을 아래에 제기한다.

(1) NFS(Network File System) NFS는 Sun Microsystems에서 개발한 열린체계로서 Unix의 가동환경들에 널리 리용되였다. 원격마디들에서 파일체계들은 단일계층에 련결(설치)된다. 프로쎄스가 파일연산요청을 내보낼 때 NFS는 그것을 원격절차호출(remote procedure call, RPC)로 변환하고 표준 TCP/IP를 통하여 원격마디로 보낸다.

파일연산은 원격마디에 의해 실행되며 그 결과는 RPC에 의해 되돌려 진다. 이것은 투명하게 진행된다. 사용자는 국부파일에 접근하는것만큼 쉽게 원격파일에 접근할수 있다. NFS의 기본문제는 약한 확장가능성이다.

(2) AFS(Andrew File System) AFS는 파일체계의 확장가능성을 높이는데 목적을 두고 있다. 여기서는 큰 분산체계를 매개가 자체의 파일봉사기를 가지는 토막들의 계층으로 나눈다. 망통신흐름을 줄이기 위하여 AFS는 대규목적인 캐쉬화를 진행하며 NFS의 상태 없는 규약대신에 상태 있는 규약을 받아 들인다.

NFS가 여러 대륙에 걸쳐 수천개의 마디들로 확장될수 있다는것이 증명되였다. 그러나 AFS를 통한 국부파일접근은 보통의 Unix파일연산들보다 더 느리다(이것은 속도가 높아 지면 단일마디의 성능이 약해 지는 병렬체계와 류사하다.).

(3) Solaris MC NFS와 AFS는 다 성숙된 상업체계들이다. 이 방법은 자원들이 리용가능하면 단일파일계층을 창조하기 위하여 워크스테이션의 단일파일계층을 워크스테이션의 파일체계로 수정한다. 다행히도 대부분의 Unix체계들은 핵심부의 원천코드가 없이 새로운 파일체계를 핵심부에 첨가할수 있게 하는 표준vnode대면부를 제공한다.

이 방법은 요구에 맞게 새로운 파일체계를 만들수 있다는 우월성을 가진다. 사실 AFS는 본래 이 방법으로 실현되였다. 부족점은 파일체계개발이 여러달 걸린다는것이다. Solaris MC는 9.4에서 론의되는 이 방법의 실례를 제공한다.

9. 3. 3. 단일 I/0와 망작업 및 기억기공간

SSI를 달성하는데 단일조종점, 단일주소공간, 단일일감관리체계, 단일사용자대면부, 단일프로쎄스조종이 요구된다.

이 개념들은 그림 9-11에서 서술된다. 이 실례에서 매 마디는 정확히 하나의 망접속을 가진다. 4개 마디들중 2개는 매개가 련결된 2개의 I/0장치를 가진다.

단일망화

적당히 설계된 클라스터는 하나의 체계처럼 동작한다. 다시 말하여 그것은 4개의 망접속과 련결된 4개의 I/0장치를 가지는 하나의 큰 워크스테이션과 같다. 임의의 마디에서

임의의 프로세스는 임의의 망과 I/0장치를 마치도 그것들이 국부마디에 련결된것처럼 사용할수 있다.

단일망화는 임의의 마디가 임의의 망접속에 접근할수 있다는것을 의미한다.

단일입력/출력

그림 9-11의 클라스터가 Web봉사기로 사용된다고 하자.

Web정보자료기지는 2개의 RAID사이에 분산된다. http데몬의 4개의 망접속으로부터 오는 Web요청들은 프로세스하는 매 마디들에서 시동된다.

단일 I/0공간은 임의의 마디가 2개의 RAID에 접근할수 있게 한다. 대부분의 요청들은 ATM망으로부터 온다고 가정한다. 마디 3의 http함수들이 4개의 마디모두로 분산될수 있다면 리로울것이다.

단일조종점

체계관리자는 전체 클라스터와 개별적인 매개 마디를 단일한 점에서 구성, 감시, 검사, 조종할수 있어야 한다.

많은 클라스터들은 그림 9-11에서와 같이 클라스터의 모든 마디들에 접속된 체계 console를 통하여 이것을 돕는다. 체계console는 체계관리자가 관리동작을 수행하기 위하여 LAN의 어디서나 체계console에 원격접속할수 있도록 외부의 LAN(그림 9-11에서는 보여 주지 않는다.)에 접속된다.

단일조종점은 모든 체계관리동작이 체계console에 의해서 단독으로 수행된다는것을 의미하지 않는다는데 주의해야 한다. 실제로 많은 관리기능들은 클라스터의 여기저기에 분산된다. 이것은 클라스터조종의 SMP나 대형콤퓨터를 관리하는것보다 힘들지 않다는것을 의미한다. 그것은 관리관련정보가 하나의 론리적장소에 보존되여야 한다는것을 말한다.

관리자는 하나의 그라픽스도구를 가지고 클라스터를 감시한다. 그라픽스도구는 클라스터에 대한 전체 형태를 보여 주며 관리자는 의사에 따라 확대 및 축소할수 있다.

단일조종점(또는 단일관리점)은 클라스터체계를 구성하는데서 가장 힘든 문제들중의 하나이다.

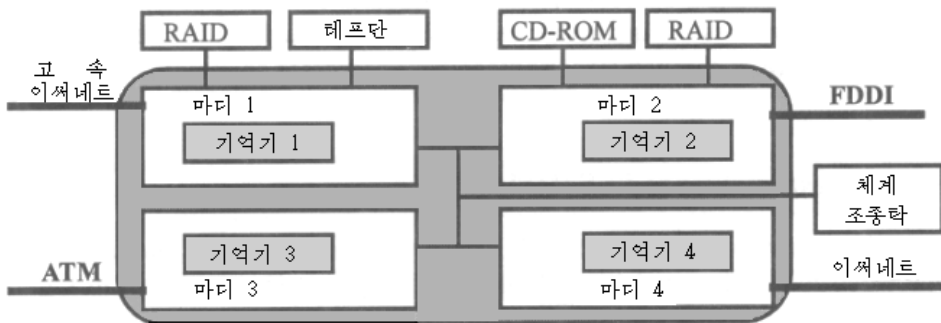



그림 9-11. 단일망화, 단일 I/O 공간, 단일기억기, 단일조종점에 대한 설명

분산 및 망체계관리의 기술들이 클라스터에 이전될수 있다.

망관리를 위한 여러개의 표준들이 이미 개발되였다. 한가지 실례는 SNMP(Simple Network Managemant Protocol)이다. 주요콤퓨터판매자들은 IBM NetView나 HP OpenView와 같은 정교한 전용망관리쏘프트웨어를 개발하였다. HP는 또한 HP클라스터관리를 위하여 ClusterView라고 하는 쏘프트웨어를 특별히 개발하였다.

그럼에도 불구하고 유용성지원체계, 파일체계, 일감관리체계와 같이 클라스터쏘프트웨어의 나머지부분과 잘 통합되는 사용이 쉽고 강력하고 효률적인 클라스터관리쏘프트웨어를 개발하는것은 여전히 중요한 연구 및 개발분야이다.

단일기억기공간

단일기억기공간은 사용자에게 하나의 크고 집중된 주기억기에 대한 환상을 준다. 실지 이것은 분산된 국부기억기들의 모임이다.

PVP와 SMP, PSM들은 이러한 관점에서 MPP와 클라스터들우에 있다. 왜냐하면 그것들은 프로그람이 모든 국부 또는 대역기억기들을 리용하게 하기때문이다.

클라스터가 단일기억기공간인가를 검사하는 좋은 방법은 임의의 단일마디가 제공할수 있는 기억기공간보다 더 큰 기억기공간을 요구하는 순차프로그람을 실행하는것이다. 그림 9-11의 매 마디는 사용자가 리용가능한 256MB의 기억기를 가진다고 하자. 리상적인 단일기억기형태는 클라스터가 1GB의 기억기를 요구하는 순차프로그람을 실행하게 한다. 이것은 클라스터를 SMP로 느끼게 한다.

클라스터우에서 단일기억기공간을 실현하기 위한 여러가지 방법들이 제기되였다. 쏘프트웨어 DSM의 방법은 이미 5장에서 론의하였다. 다른 방법은 콤파일러가 응용의 자료구조들을 여러 마디들로 분산시키는것이다.

효률적이고 가동환경독립이며 순차2진코드를 지원할수 있는 단일기억기구조를 개발하는것은 여전히 힘든 과제이다.

그밖의 단일형태

단일체계영상의 최종목표는 워크스테이션이나 SMP처럼 클라스터를 리용하기 쉽게 하는것이다. 아래에 워크스테이션들에서 제기된 보충적인 단일체계영상들을 서술한다.

- **단일일감관리체계** 모든 클라스터일감들은 임의의 마디로부터 단일일감관리체계로 제출될수 있다. 9.5에서 론의하는바와 같이 오늘 리용가능한 그러한 체계들은 많다.
- **단일사용자대면부** 사용자들은 단일그라픽스대면부를 통하여 클라스터를 리용할수 있어야 한다. 그러한 대면부는 워크스테이션(실례로 Common Desktop Environment 또는 CDE)들과 PC들(실례 Microsoft Windows 95)에 리용가능하다. 클라스터의 개발에 참가하는 좋은 방향은 Web기술을 리용하는것이다.
- **단일프로쎄스공간** 사용자의 모든 프로쎄스들은 그것들이 상주하는 그 어디에서나 단일한 프로쎄스공간에 소속되며 단일한 프로쎄스식별자구조를 공유한다. 임의의 마디의 프로쎄스는 원격마디의 프로쎄스들을 창조(실례로 Unix의 fork를 통하여)하거나 그것들과 통신(실례로 신호, 관흐름 등)할수 있다.

9. 4. Solaris MC의 단일체계영상

Sun Microsystems는 단일마디Solaris핵심부[369]의 견본확장인 Solaris MC(MC는 multicomputer를 의미한다.)를 개발하였다. 이것을 그림 9-12에서 보여 준다. 마디들은 Solaris 조작체계를 실행하는 Sun워크스테이션들에 대응한다.

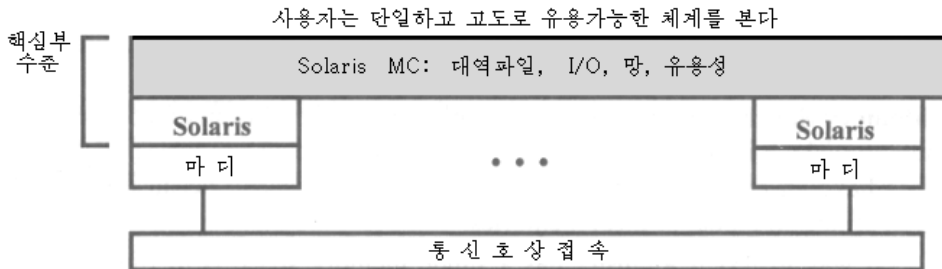



그림 9-12. Sun Microsystems의 Solaris MC에 대한 개념도

Solaris MC는 핵심부수준에서 단일체계영상과 높은 유용성을 제공하며 핵심부우의 다른 방법들보다 더 효률적인것으로 인정되고 있다.

Solaris MC는 대상지향기술을 통하여 실현된다. 또한 이것은 대상지향언어 C++, 표준COBRA(Common Object Request Broker Architecture)대상모형 및 그 대면부정의언어(IDL)를 널리 사용한다.

아래에서는 Solaris MC가 어떻게 단일프로쎄스공간과 단일망화, 단일I/O공간을 실현하는가를 론의한다.

9. 4. 1. 대역파일체계

Solaris MC는 Proxy파일체계(PXFS)라고 하는 대역파일체계를 사용한다. 주장하는 기본특징은 단일체계영상과 일관성의미가 높은 성능이다. PXFS를 그림 9-13에서 설명한다.

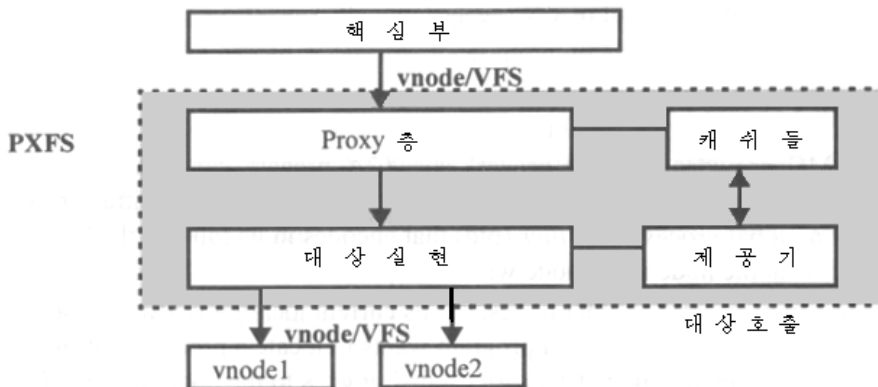



그림 9-13. Solaris MC의 Proxy 파일체계

Proxy파일체계

PXFS는 파일접근이 프로쎄스와 파일위치에 대하여 투명하게 한다. 즉 임의의 프로쎄스는 같은 경로이름을 가지고 임의의 마디들의 임의의 파일에 접근할수 있다. 이것은 클라스터의 파일체계를 워크스테이션의 파일체계와 꼭 같게 한다. PXFS는 단일체계영상을 Vnode/UFS대면부의 파일연산들을 가로채서 달성한다.

현대 Unix체계들(Solaris 포함)은 가상파일체계(VFS)를 통하여 파일연산들을 수행한다. 여기서 Vnod는 등록부, 기호적련결, 특수한 장치, 흐름(Stream), 교환파일(swap file) 등과 같다. 이것은 보통의 자료 및 코드파일들에 대한 추상이다. 워크스테이션에서 Solaris의 핵심부는 VFS/Vnode의 연산들을 통하여 파일에 접근한다.

클라스터에서 프로쎄스와 파일은 서로 다른 마디에 상주할수 있다. 의뢰기마디가 VFS/Vnode연산을 수행하면 Solaris MC의 proxy층은 먼저 VFS/Vnod연산을 대상호출로 전환하여 파일이 상주하는 마디(봉사기마디)로 보낸다. 그다음 호출된 대상은 봉사기마디의 Solaris파일체계에서 국부VFS/Vnode연산을 수행한다. 이 실현방법은 Solaris나 파일체계의 수정을 요구하지 않는다.

파일캐쉬화

성능을 개선하기 위하여 PXFS는 원격대상호출을 줄이기 위하여 의뢰기의 확장캐쉬를 사용한다. 캐쉬화대상은 캐쉬자료를 관리하도록 의뢰기에 보존되며 봉사기의 캐쉬화대상은 파일의 의미적일관성을 유지한다. 파일의 페지들은 봉사기마디의 안정기억과 하나이상의 의뢰기마디의 캐쉬에 상주한다. PXFS는 령복사, 마디들사이의 대규모자료전송을 수행하기 위하여 bulkio대상조종기(handler)를 가진다.

PXFS는 페지가 여러 마디들을 위한 읽기용캐쉬로 또는 단일한 마디를 위한 읽기쓰기캐쉬로 되도록 하는 통표식일관성규약을 사용한다.

프로쎄스가 공유페지에 쓰려면 먼저 통표를 획득해야 한다. 이 불결한 페지를 다른 마디로 보내면 갱신내용을 잃지 않도록 먼저 봉사기마디의 안정기억에 쓴다. 통표는 읽기/쓰기체계호출의 원자성을 실시하는데도 사용된다.

9. 4. 2. 대역프로쎄스관리

Solaris MC는 단일프로쎄스공간을 제공한다.

프로쎄스는 프로쎄스의 모든 스레드들이 같은 마디에 상주해야 함에도 불구하고 임의의 마디에 상주할수 있다. 임의의 프로쎄스는 최상위의 비트들에 프로쎄스의 원천마디를 부호화한 대역프로쎄스식별자(pid)에 의해 배치된다.

원천마디는 프로쎄스가 창조된(태여 난) 마디이다. 프로쎄스는 다른 마디에로 이동할수 있는데 프로쎄스의 현재위치는 항상 자기의 원천마디에 보존된다. 사용자나 체계프로그람이 프로쎄스 P를 찾으려면 먼저 P의 pid를 가지고 국부캐쉬를 조사한다. P가 꺼내지지 않으면 P의 원천마디로 간다.

분산마디관리자

Solaris MC는 그림 9-14에서 설명하는바와 같이 Solaris핵심부층의 최상위에 대역프로쎄스층을 첨가하여 단일프로쎄스공간을 실현한다.

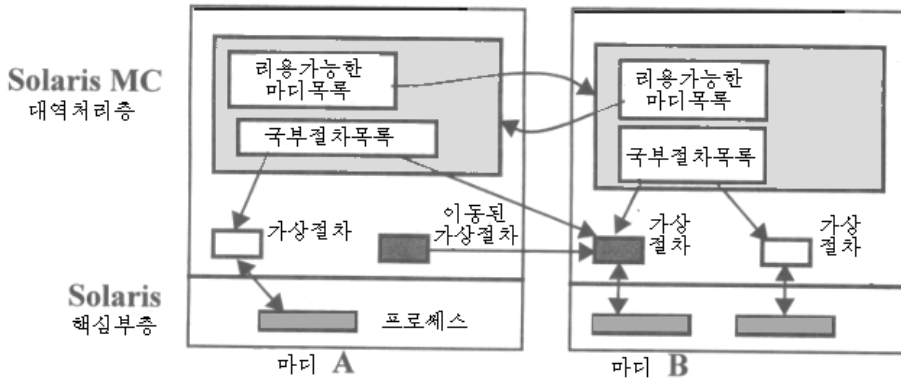



그림 9-14. Solaris MC의 프로쎄스관리에 사용된 자료구조

매 마디에 하나의 마디관리자가 있으며 매 국부프로쎄스에 하나의 가상프로쎄스(vproc)대상이 있다. vproc는 매 프로쎄스의 부모와 자식에 대한 정보를 보존한다.

마디관리자는 2개의 목록 즉 리용가능한 마디의 목록과 국부프로쎄스의 목록을 보존하며 이동한 마디목록들을 포함한다. 프로쎄스가 다른 마디로 이동할 때 대리자vproc는 계속 원천마디에 보존된다. 대리자vproc에 의해 수신된 연산들은 프로쎄스가 상주하는 현재의 마디로 보내 진다.

대역프로쎄스의 관리를 실현하려면 추가적인 작업이 진행되여야 한다.

실례로 모든 프로쎄스관련체계호출들은 대역프로쎄스층에로 방향이 바뀌고 일부 hook들이 대역층을 호출하는 Solaris핵심부에 첨가되며(이것을 핵심부의 수정을 의미한다.) 서로 다른 마디들의 대역층들은 IDL대면부로 통신할것을 요구한다. 국부proc파일체계들은 ps와 같은 Unix연산들이 하나의 대역proc를 조사하여 모든 국부Vnod/VFS대면부와 같은 열린 표준대면부의 부족으로 힘들게 실현된다는것을 발견하였다. 그러한 표준의 구성은 클라스터체계를 지원하는 단일프로쎄스공간의 개발을 크게 촉진proc에 접근할수 있도록 하나의 대역proc로 통합된다.

9. 4. 3. 단일 I/O체계영상

클라스터의 I/O부분체계들에서 단일형태를 지원하기 위한 2가지 기술이 개발되였다. 하나는 통일적인 장치이름짓기이며 다른것은 단일망이다.

통일적인 장치이름짓기

Solaris MC는 통일적인 장치이름을 가지는 단일I/O부분체계영상을 제공한다. 장치번호는 장치형태와 기판의 수와 마찬가지로 그 장치의 마디번호로 구성된다. 프로쎄스는 이 이

름을 사용하여 그것이 원격마디에 련결되였을 때조차도 마치 국부마디에 련결된것처럼 임의의 장치에 접근할수 있다.

장치구동프로그람들은 동적으로 적재가능하며 동적으로 구성가능하다. 구성은 분산된 장치의 봉사기에 의하여 그대로 보존되며 새로운 장치가 클라스터의 임의의 마디로 첨가될 때마다 통지된다. 임의의 마디의 프로쎄스가 장치구동프로그람을 호출하면 구동프로그람은 그 마디에 적게 된다.

Solaris MC는 Solaris의 모든 장치구동프로그람들이 변화없이 실행되도록 실현된다.

단일망

Solaris MC는 응용들이 실행되는 어느 마디에 상관없이 현재의 망응용들이 수정되지 않고 같은 망접속성을 가진다는것을 담보한다. 망구조를 그림 9-15에서 설명한다.

단일워크스테이션과 꼭같이 Solaris MC클라스터는 여러 마디들에 련결된 여러개의 망장치들(실례로 개개가 이써네트, 고속이써네트, ATM, FDDI 등을 위한것)을 가진다. 그러나 임의의 프로쎄스는 임의의 망장치를 마치 그것이 국부마디에 련결된것처럼 사용할수 있다.

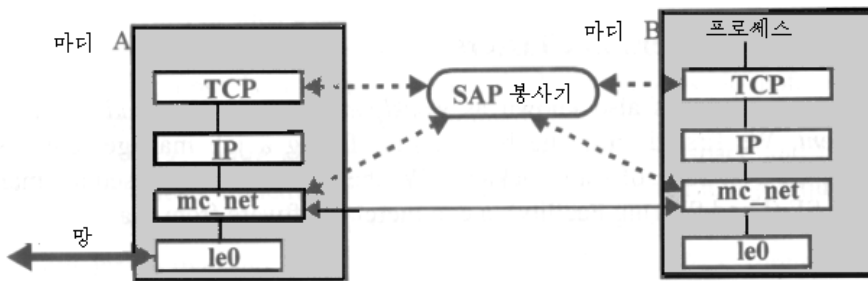



그림 9-15. Solaris MC의 망구조(굵은선: 자료통신, 점선: 조종통신, SAP: 봉사접근점)

모든 프로쎄스들은 임의의 마디로부터 망에 대한 같은 보기를 가진다. 망은 망장치 le0을 통하여 오직 마디 A에만 물리적으로 접속된다. 그러나 마디 B의 프로쎄스는 le0를 마치도 그것이 마디 B에 물리적으로 련결된것처럼 망에 접근하는데 사용할수 있다. 프로쎄스가 통보문을 보내려고 할 때 그것은 다음과 같이 달성된다.

(1) 모든 규약프로쎄스(실례로 TCP/IP)는 국부마디(즉 마디B)에서 진행된다.
(2) 마디 B의 mc-net모듈은 le0의 위치를 찾아 마디 A의 mc-net에로 파케트를 보낸다.
(3) 마디 A의 mc-net는 파케트를 물리적망으로 내보내는 국부망연산들을 수행한다.

봉사접근점

망봉사들은 봉사접근점(service access point, SAP)봉사기를 통하여 접근한다. TCP/IP에서 하나의 SAP는 바로 하나의 포구이다. SAP이름공간자료기지는 SAP봉사기에서 유지되며 클라스터의 하나이상의 마디들에 제공된다. 모든 프로쎄스들은 SAP가 어느 마디에

있는가를 알기 위하여 SAP봉사기로 간다. SAP봉사기는 또한 같은 SAP가 동시에 서로 다른 마디들에 한정되지 않는다는것을 담보한다.

Solaris Mc는 여러 마디들이 망봉사를 위해 사본된 SAP봉사기처럼 동작하도록 한다. 실례로 rlgin과 telnet, http봉사기들은 모든 망에 사본된 기정값이다. 사용자는 매 봉사기에 대하여 하나의 단일입력자료를 알고 있다.

Solaris Mc클라스터가 Web봉사기로 사용된다고 하자. 이때 사용자가 단일URL을 사용하여 클라스터(마디는 아니다.)에 접근하면 체계는 사용자를 가장 바쁘지 않은 마디의 http봉사기에 자동적으로 접속한다. 이 방법으로 모든 마디들은 http요청들에 병렬로 봉사할수 있으며 이것은 처리량을 높이고 응답시간을 줄인다.

9. 5. 클라스터에서의 일감관리

일감관리는 작업부하관리 또는 부하공유, 부하관리로도 알려 져 있다.

먼저 일감관리체계에서 부닥치는 기본문제들을 론의하고 리용가능한 쏘프트웨어묶음들을 개괄한다. 다음 널리 사용되는 일감관리체계 LSF(Load Sharing Facility)와 가동환경 Computing의 상업쏘프트웨어를 자세히 설명한다.

9. 5. 1. 일감관리체계

일감관리체계(Job Management System, JMS)는 3개의 부분을 가진다.

- 사용자봉사기는 사용자가 하나이상의 대기렬에 일감들을 제출하고 매 일감에 대한 자원요구를 서술하며 대기렬로부터 일감을 지우고 일감이나 대기렬의 상태를 요구하게 한다.
- 일감일정짜기는 일감형태와 자원요구, 자원유용성, 일정짜기책략에 따라 일감의 일정짜기와 대기렬만들기를 수행한다.
- 자원관리자는 자원들을 배정하고 감시하며 일정짜기책략을 실시하고 결정정보들을 수집한다.

그밖의 기능들에는 일감이동과 검사점설정이 있다.

JMS 관리와 사용

JMS의 기능은 흔히 분산적으로 실현된다. 실례로 사용자봉사기는 매 호스트마디에 상주하고 자원관리자는 클라스터의 모든 마디들에 퍼질수 있다.

그러나 JMS의 관리는 집중식이다. 모든 구성 및 기록파일들은 한 위치에서 유지되여야 한다. 또한 JMS를 사용하기 위한 단일사용자대면부가 있어야 한다.

사용자가 한 쏘프트웨어를 통하여 PVM일감들을 실행하고 다른 쏘프트웨어를 통하여 MDI일감들을 실행하며 또 다른 쏘프트웨어를 통하여 HDF일감들을 수행하게 하는것은 좋지 않다.

PVM과 MPI, HPF는 4편의 프로그람작성장들에서 론의된다.

JMS는 실행하는 일감들에 최소한의 영향을 주면서 클라스터를 동적으로 재구성할수 있어야 한다.

관리자의 서문 및 종문스크립트들은 보안검사와 계정, 지우기를 위하여 매 일감의 전과 후에 실행될수 있어야 한다.

사용자들은 자기의 일감들을 깨끗이 지울수 있어야 한다.

관리자나 JMS는 임의의 일감을 깨끗히 유지하거나 지울수 있어야 한다. 깨끗이라는 단어는 일감이 유지되거나 지워 질 때 그것의 모든 프로쎄스들이 포함되여야 한다는것을 의미한다. 그렇지 않으면 일부 《고아》프로쎄스들이 체계에 남아서 클라스터의 자원을 소비하며 종당에는 체계를 리용할수 없게 만든다. NASA의 사용자들은 고아프로쎄스들과 외래일감들을 찾고 지우는 능력의 부족이 현재의 JMS에서 유일하게 가장 큰 문제라는것을 인정하였다.

일감형태

여러 형태의 일감들은 하나의 클라스터에서 실행한다.

순차일감들은 단일한 마디에서 실행한다.

병렬일감들은 여러 마디를 사용한다.

호상작용일감들은 빠른 회전시간을 요구하는 일감들이며 그것들의 입력/출력은 하나의 말단으로 향한다.

이 일감들은 큰 자원을 요구하지 않으며 사용자는 그것들이 즉시에 실행하기를 기대하며 대기렬에서 기다리게 하지 않는다.

묶음일감들은 보통 큰 기억기공간과 긴 CPU시간과 같은 더 많은 자원들을 요구한다. 그러나 그것들은 즉시적인 응답을 요구하지 않는다. 그것들은 자원이 리용가능하게 될 때 실행을 위하여 일정이 짜지도록 일감대기렬에 제출된다.

호상작용 및 묶음일감들은 JMS에 의하여 관리되며 외래일감은 JMS의 밖에서 창조된다. 실례로 워크스테이션들의 망이 클라스터로 사용될 때 사용자는 호상작용 또는 묶음일감들을 JMS에 제출할수 있다. 한편 워크스테이션의 소유자는 임의의 시각에 JMS를 통하여 제출되지 않는 외래일감을 시작할수 있다.

그러한 일감은 클라스터의 JMS를 통하여 제출되는 클라스터일감들(호상작용 또는 묶음, 순차일감)과 대립되는 국부일감이라고도 불리운다. 국부일감의 특징은 빠른 응답시간이다.

소유자는 워크스테이션의 모든 자원들을 마치도 클라스터일감들이 존재하지 않는것처럼 자기의 일감수행에 리용하려 한다.

클라스터작업부하의 특징

일감관리문제들을 실제적으로 처리하자면 클라스터작업부하의 형태를 리해하여야 한다. 실제의 클라스터에서 긴 시간의 연산자료에 기초하여 작업부하를 특징 짓는것이 리상적이다. 그러나 그것은 극히 힘들다.

Arpaci et al.[44]는 Berkeley Now과제에서 다음의 방법을 사용하였다. 그들은 53개

워크스테이션들의 망으로부터 2달동안의 실제의 순차작업부하기록들을 모으고 32개 마디 CM−5로부터 1달동안의 병렬작업부하기록들을 수집하였다.

병렬작업부하기록들은 일감의 개발 및 생성을 포함한다. 그다음 이 기록들은 서로 다른 순차 및 병렬작업부하결합과 자원배정, 일정짜기책략에 기초하여 여러가지 통계적 및 성능값들을 생성하기 위한 모의기에 입력된다.

Arpaci의 연구는 다른 연구자들의 측정결과가 일치하는 다음의 작업부하특징들을 보여 준다.

- 병렬일감들의 거의 절반은 정규적인 작업시간에 제출된다.
- 약 80%의 병렬일감들은 3분동안 또는 그이하에서 실행된다.
- 90분동안 또는 그이상 실행하는 병렬일감은 전체 실행시간의 50%이상을 차지한다.
- 순차작업부하는 워크스테이션의 60%～70%가 임의의 시각에 지어 낮시간에도 병렬일감을 실행할수 있다는것을 보여 준다.
- 워크스테이션에서 모든 휴식주기의 53%는 3분이하이지만 휴식시간의 95%는 10분 또는 그이상의 시간주기들에서 소비된다.
- 2:1의 규칙이 적용된다.

이것은 적당한 JMS쏘프트웨어를 가지는 64개 워크스테이션들의 망은 보통의 순차작업부하이외에 32개 마디병렬작업부하를 유지한다는것을 의미한다. 다시 말하여 클라스터는 클라스터크기절반을 초고속콤퓨터에 자유로 준다.

일감일정짜기

일감일정짜기문제는 2.2.6에서 론의된 프로쩨스의 일정짜기와 류사하다. 그밖의 일감들은 특정한 시간(달력일정짜기)에 또는 특정한 사건이 일어 날 때(사건일정짜기) 실행을 위해 일정이 짜진다.

클라스터에서 일감일정짜기문제를 풀기 위한 여러가지 방법들을 표 9−3에 제시한다. 일감들은 복종시간과 자원(마디, 실행시간, 기억기, 디스크 등)요구, 일감형태, 사용자의 신원에 기초하여 계산되는 우선권에 따라 일정이 짜진다.

정적우선권에서 일감들은 미리 결정된 고정된 방법에 따라 우선권이 배정된다. 단순한 방법은 먼저 온것을 먼저 봉사하는 형태로 일감들의 일정을 짜는것이다.

다른 방법은 서로 다른 사용자에게 서로 다른 우선권을 배정하는것이다. 동적우선권에서 일감의 우선권은 시간에 따라 변한다. 흔히 사용되는 방법은 낮에는 회전시간을 줄이기 위하여 짧고 호상작용적인 일감들에 더 높은 우선권을 배정하며 저녁시간동안에는 체계리용을 개선하기 위하여 타일붙이기술을 사용하는것이다.

일감의 자원요구는 정적 또는 동적일수 있다.

실례로 일감에 필요한 마디의 수는 고정될수 있다. 즉 같은 마디들은 일감실행동안에 휴식상태가 되여도 그것의 전체 존재기간에 일감에 배정된다.

현대 클라스터들에서는 정적자원이 널리 사용된다. 그러나 그것은 클라스터의 자원리용저하를 가져 오며 사용자가 부하평형을 맞추도록 요구한다. 더우기 워크스테이션의 소유

표 9-3 클라스터마디들에 대한 일감 일정짜기의 문제

문제	방법	주요문제들
일감우선권	비선점	높은 우선권일감들의 지연
	선점	부가처리, 실현
자원요구	정적	부하비평형
	동적	부가처리, 실현
자원공유	전용	낮은 리용
	공간공유	타일붙이기, 큰 일감
	시간공유	프로쎄스에 기초한 일감조종, 문맥절환부가처리
일정짜기	독립적인 일정짜기	엄중한 속도저하
	무리일정짜기	실현곤난
외래(국부)일감들과의 경쟁	머무르기	국부일감속도저하
	이동	이동턱값, 이동의 부가처리

자가 기계를 닫을 때와 같이 필요한 마디들이 리용불가능하게 되는 상황을 처리할수 없다.

동적자원은 일감이 실행시에 마디들을 획득 또는 해제하도록 한다. 그러나 그것은 실행하는 일감과 JMS사이의 협조를 요구하므로 실현하기 더 힘들다.

일감은 자원들을 첨가/지우기 위하여 JMS에로의 비동기적인 요청들을 만들것을 요구하며 JMS는 자원들이 리용가능하게 되면 일감을 통지할것을 요구한다.

동기는 일감이 요청/통지에 의해 지연(차단)되지 않는다는것을 의미한다.

일감들과 JMS사이의 협조는 현존 프로그람작성언어들과 서고들에 대한 수정을 요구한다. 그러한 협조를 위한 기본기구가 PVM에 존재하며 MPI에 대해서는 일부작업이 진행중에 있다.

전용방식

클라스터의 마디들을 공유하는데 3가지 방법들이 사용된다.

전용방식에서는 한번에 하나의 일감만이 클라스터에서 실행할수 있으며 한번에 그 일감의 최대로 1개의 프로쎄스가 하나의 마디에 배정된다. 단일일감은 그것이 다른 일감들을 실행하도록 클라스터를 해제하기전에는 완성될 때까지 실행한다.

지어 전용방식에서도 일부 마디들은 체계사용을 위하여 예약되며 사용자의 일감에는 열리지 않는다는것에 주의해야 한다. 그밖의 모든 클라스터자원들은 단일일감을 실행하는데 리용된다. 이것은 낮은 체계리용률을 가져 온다.

공간공유

공간공유방식에서 여러 일감들은 마디들의 분리된 구획들에서 동시에 실행할수 있다. 한번에 최대로 하나의 프로쎄스가 하나의 마디에 배정된다.

마디들의 구획이 하나의 일감에 전용됨에도 불구하고 호상접속 및 I/O부분체계는 모든 일감들에 공유된다.

공간공유는 다음의 실례에서 보여 주는 타일붙이기문제와 큰 일감문제를 해결해야

한다.

실례 9.13. 클라스터마디들의 타일붙이기에 의한 일감일정짜기

타일붙이기기술은 그림 9-16에서 설명된다.

그림 9-16 ㄱ)에서 JMS는 4개마디클라스터에서 먼저 온것은 먼저 봉사하는 간단한 형식으로 4개 일감을 일정 짠다. 일감 1과 2는 순차적이며 매개가 하나의 마디만을 요구한다. 그것들은 마디 1과 마디 2에 배정된다. 일감 3과 4는 병렬이며 매개가 3개의 마디를 요구한다. 일감 3이 오면 그것은 즉시적으로 실행할수 없다. 그것은 일감 2가 끝나고 필요한 마디들은 해제할 때까지 기다려야 한다.

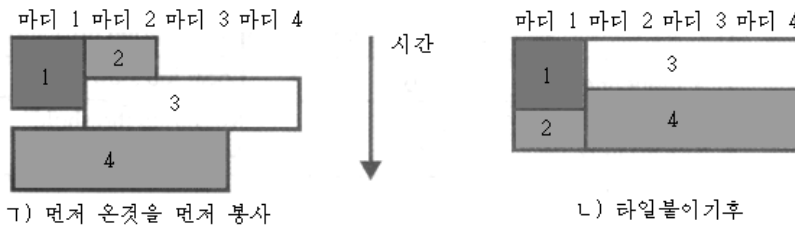



그림 9-16. 일감들을 클라스터의 마디들로 일정짜기 위한 타일붙이기기술

타일붙이기는 그림 9-16 ㄴ)에서 보여 주는바와 같이 마디의 리용률을 높인다. 4개의 일감들에 대한 총 실행시간은 일감들을 리용가능한 마디들 전면에 걸쳐 재배치함으로써 크게 감소된다.

큰 일감문제는 큰 일감이 너무 오래동안 작은 일감들의 클라스터자원리용을 막을 때 생긴다. 이 문제는 전용 또는 공간공유마디들에서 해결될수 없다.

시간공유

전용 또는 공간공유모형에서는 사용자의 하나의 프로쎄스만이 하나의 마디에 배정된다. 그러나 체계의 프로쎄스들 또는 데몬들은 여전히 같은 마디에서 실행하고 있다.

시간공유방식에서는 사용자의 여러개의 프로쎄스들이 같은 마디에 배정된다. 시간공유는 다음의 병렬일정짜기방책을 도입한다.

(1) **독립적인 일정짜기** 시간공유의 가장 간단한 실현은 전통적인 워크스테이션에서와 같이 서로 다른 프로쎄스들을 일정 짜는데 매 클라스터마디의 조작체계를 사용하는것이다. 이것은 **국부일정짜기** 또는 **독립적인 일정짜기**라고 한다.

그러나 병렬일감들의 성능은 크게 낮아 진다. 원인은 병렬일감의 프로쎄스들이 호상작용을 요구하기때문이다.

실례로 한 프로쎄스가 다른 프로쎄스와 장벽동기화를 요구할 때 다른 프로쎄스는 일정짜기가 되여 있을수 있다. 그래서 첫번째 프로쎄스는 기다려야 한다. 두번째 프로쎄스가 일정 짜지면 첫번째 프로쎄스가 교체되여 버린다.

(2) **무리일정짜기** 무리일정짜기방법은 병렬일감의 모든 프로쎄스들을 함께 일정 짠다. 한 프로쎄스가 활성이면 모든 프로쎄스들은 활성이다.

Berkeley의 연구는 무리일정짜기가 성능을 1:2로 개선할수 있다는것을 보여 준다.

클라스터의 마디들은 완전히 박자동기화되지 않는다. 사실 대부분의 클라스터들은 비동기체계들이며 같은 박자로 움직이지 않는다.

《모든 프로쎄스가 동시에 실행되도록 일정 짜진다》고 할 때 그것들은 정확히 같은 시간에 시작하지 않는다. 무리일정짜기삐뚤어짐은 첫번째 프로쎄스가 시작한 시간과 마지막 프로쎄스가 시작한 시간사이의 최대차이다.

Berkeley의 연구는 병렬일감의 실행구간은 무리일정짜기삐뚤어짐이 더 커짐에 따라 증가하여 더 긴 실행시간을 가지게 된다는것을 보여 준다.

NOW는 더 큰 시간토막(1s) 또는 작은 무리일정짜기삐뚤어짐(10ms)을 가진다.

Berkeley연구는 무리일정짜기가 효과적인 동질클라스터를 사용하였다. 그러나 무리일정짜기는 실현의 곤난성으로 하여 아직 대부분의 클라스터들에서 실현되지 못하였다.

(3) **외래(국부)일감들과의 경쟁** 클라스터의 일감들과 국부일감들이 다 실행하면 일정짜기는 더 복잡해 진다. 국부일감들은 클라스터의 일감들보다 높은 우선권을 가진다. 소유자는 한번의 건반누르기로 모든 워크스테이션자원들을 조종하고 싶어 한다.

그러한 상황을 프로쎄스하는데 기본적으로 2가지 방법이 있다.

클라스터의 일감은 워크스테이션마디에 머무르던가 다른 휴식마디로 이동할수 있다. 머무르기방법은 이동비용을 없앤다는 우월성을 가진다. 여러 머무르기방법들이 제안되였다.

실례로 클라스터의 프로쎄스는 가장 낮은 우선권으로 실행될수 있다. 워크스테이션의 주기들은 3개의 부분 즉 핵심부의 프로쎄스, 국부프로쎄스, 클라스터프로쎄스를 위한 부분으로 나눌수 있다[213]. 그러나 특히 클라스터의 일감이 빈번한 동기화와 통신을 요구하는 부하평형병렬일감인 경우 머무르기는 국부 및 클라스터의 일감들의 속도를 낮춘다. 이것은 주로 작업부하의 평형을 위하여 일감들이 리용가능한 마디들로 흐르는 이동방법을 리용하게 된다.

이동방법

이동방법은 다음의 3가지 문제점을 고려해야 한다.

- **마디유용성** 일감이 이동해야 할 다른 리용가능한 마디를 찾을수 있는가? Berkeley의 연구는 대답이 거의 긍정적이라는것을 가리킨다. 가장 많이 사용되는 시간에도 클라스터의 60%의 워크스테이션들은 리용가능하기때문이다.
- **이동의 부가처리** 이동의 부가처리의 영향은 무엇인가? 이동시간은 병렬일감의 속도를 크게 낮출수 있다. Berkeley의 연구는 속도낮춤이 2.4배로써 크다는것을 가리키고 있다. 그러므로 이동부가처리를(실례로 통신부분체계를 개선하며) 줄이든가 이동을 극히 제한하는것이 중요하다.

속도낮춤은 병렬일감이 2배 크기의 클라스터에서 실행할 때보다는 작다. 실례로 32개마디병렬일감이 60개마디클라스터에서 실행될 때 이동에 의한 속도낮춤은 이동시간이 3분으로 길 때조차도 20%이하이다. 이것은 더 많은 마디들이 리용가능하여 이동에 대한 요구가 적어 지기때문이다.

- **보충턱값** 보충턱값은 얼마인가? 가장 나쁜 경우는 프로쎄스의 소유자가 곧 요구하게 될 바로 그 마디를 이동할 때이다. 그러므로 프로쎄스는 다시 이동해야 하며 주기는 연기된다. 보충턱값은 클라스터가 워크스테이션을 휴식마디로 간주하기전까지 워크스테이션이 사용되지 않고 머무르는 시간의 량이다.

Berkeley연구는 3분의 보충턱값이 병렬일감의 성능을 최대화한다는것을 보여 준다.

일정짜기방책의 명세서

사용자의 자원요구는 복잡하므로 JMS는 복잡한 일정짜기규칙들을 실현하는 융통성있는 일정짜기대면부를 가져야 한다. 선택들에 대한 안내는 융통성이 충분하지 않다. 대면부는 또한 대부분의 사용자들에게 알맞는 기정의 일정짜기방책을 제공해야 한다.

9. 5. 2. 일감관리체계들에 대한 개괄

상업 및 공공령역에서 현재 20개이상의 JMS묶음(package)들이 사용되고 있다. Berker et al.[56]은 잘 정의된 기준들의 모임에 따라 이 묶음들을 평가하였다.

표 9-4는 그러한 기준들에 따라 분류한 이 묶음들의 최상위 5개를 개괄한다. Web자원은 이 모든 묶음들에 대한 지시자들을 포함한다.

DQS 공공령역묶음들중의 하나인 DOS(Distributed Queueing System)는 더욱 완성되고 있으며(현재 판본 3) 지금 널리 사용되고 있다. DQS는 Florida State대학에 의해 창조되며 유지되고 있다. 또한 독일의 GENIAS GmbH에 의해 제공된 Codine이라고 부르는 상업용판본을 가지고 있다. GENIAS는 Codine이 사실상의 유럽표준으로 되였다고 주장하고 있다.

LSF 전 세계에 20000이상의 사용허가를 가지는 가동환경 Computing Corporation의 LSF(Load Sharing Facility)는 대체로 가장 널리 사용된다. 9.5.3에서 LSF를 상세히 론의한다.

NQS 2개의 다른 묶음 즉 Codor와 NQS(Network Queueing System)가 있으며 표 9-4에는 기록되지 않았지만 대단히 영향력이 있다. Codor는 Wisonsin대학에서 시작된 공공령역쏘프트웨어묶음이다. 그것은 휴식워크스테이션의 주기들을 리용하며 검사점과 프로쎄스의 이동을 지원하는 첫번째 체계들중의 하나이다.

NQS는 Stirling Software에 의해 이미 1980년대에 시장에 나왔는데 DEC와 IBM, Cray의 연구용초고속콤퓨터들의 NAS and box용JMS를 생산하기 위하여 NASA Ames Research laboratory와 계약하였다.

Connect : Queue NQS는 사실상의 JMS표준으로 되였으며 묶음인자는 프로쎄스(processing)를 위하여 여러가지 초고속콤퓨터들에서 아직도 널리 사용되고 있다.

현재의 대부분의 JMS묶음들은 NQS호환이라고 부르는데 이것은 그것들이 NQS지령들을 지원한다는것을 의미한다. 상업NQS는 Sterling Software의 Connect:Queue로 발전하였다. Generic NQS라는 공공령역NQS의 실현도 있는데 영국의 Sheffield대학에서 유지되고 있다.

표 9-4의 묶음들은 다음의 특징들을 가진다. 특징들중 일부는 표에 명확히 기록되지 않았다. 이 특징들은 다른 묶음들에 대해서도 전형적이다. 이 특징들중 일부는 9.5.3에서 LSF를 론의할 때 자세히 본다.

- 사용자지원이 제공되는데 상업묶음들은 일반적으로 더 좋은 사용자지원을 제공한다.
- 묶음들은 모두 이질클라스터들을 지원한다. 매 마디는 주요Unix가동환경들중 임의의것이 될수 있다. 그러나 Windows NT, Windows 95, Windows 3은 지원하지 않는다.
- 묶음들은 모두 여러개의 설정가능한 형태의 일감대기렬을 제공한다.
- 이 묶음들이 기업클라스터에서 사용될 때 JMS로 관리되는 클라스터의 일감들은 국부일감들을 수행하는 워크스테이션의 소유자에게 영향을 준다. 그러나 NQE와 PBS는 그 영향을 조절할수 있다. DQS에서는 그 영향이 최소로 되도록 설정할수 있다.

표 9-4 일감관리체계들에 대한 개괄

JMS이름	상업용묶음들			공공령역묶음들	
	Connect: Queue	LSF	NQE	DQS	PBS
판매자/창조자	Sterling	가동환경	Cray	FSO	NASA
가동환경(Unix)	예	예	예	예	예
묶음일감지원	아니	예	예	예	예
호상작용일감지원	예	예	예	예	예
병렬일감지원	예	예	예	예	예
소유자에 대한 영향	예	예	가변	작다	가변
부하평형	예	예	예	예	예
검사점	아니	예	예	예	아니
프로쎄스의 이동	아니	아니	아니	아니	아니
일감감시	예	예	예	예	예
유지/다시시작	아니	예	예	예	예
동적자원	예	예	예	예	예
사용자대면부	GUI	GUI	GUI,WWW	GUI	지령행
고장허용	예	예	일부	예	약간
보안	Unix	Kerboros	US DoD	Kerboros	Kerboros

- 모든 묶음들은 클라스터의 제품들을 효과적으로 리용하기 위한 어떤 종류의 부하평형기구를 제공한다.
- 일부 묶음들은 검사점을 지원한다.

- 표 9-4의 묶음들은 동적프로쎄스이동을 지원하지 않는다. 그것들은 정적이동을 지원한다. 프로쎄스는 처음으로 창조될 때 원격마디에서 실행되도록 보내질수 있다. 그러나 그것이 일단 실행을 시작하면 그 마디에 머무른다. 동적프로쎄스이동을 지원하는 묶음은 Conder이다.
- 모든 묶음들은 사용자 또는 관리자에 의한 사용자일감의 동적인 유지 및 재시작을 가능하게 한다.
- 모든 묶음들은 자원(실례로 마디)들을 자원저장소에 동적으로 첨가하거나 자원저장소로부터 동적으로 제거하게 한다.
- 대부분의 묶음들은 지령행대면부와 도형사용자대면부를 다 제공한다. NQE는 지어 Web와 비슷한 사용자대변부도 제공한다.
- 대부분의 묶음들은 Unix보안기구들외에 Kerboros인증체계를 사용한다.

9. 5. 3. 부하공유기능(LSF)

LSF는 가동환경Computing의 상품화된작업부하관리체계이다. 그것은 Tronto대학에서 개발한 Utopia체계[668]로부터 발전되였다.

LSF는 검사점설정, 유용성, 부하이동, 단일체계영상에 대한 지원을 가진다. LSF는 고도로 확장가능하여 수천개 마디로 된 클라스터를 지원할수 있다. LSF는 PC들과 워크스테이션들로부터 IBM SP2 MPP들까지의 여러가지 Unix 및 Windows /NT가동환경들을 위하여 실현되였다. 최근에 LSF는 광대역망을 지원하는데로 확장되였다.

리론적으로 LSF는 여러 인터네트싸이트에서 실행하는 일감들을 관리하는데 사용될수 있다.

LSF	응 용 들	모든 사용자프로그람들과 명령들						
	유틸리티들	lstools	lsbatch	lstcsh	lsmake	PVM	GUI	…
	API	LSLIB(부하공유서고)						
	봉사기데몬들	LIM			RES			
	조 작 체 계	Unix 가동환경 : AIX, HP-UX, IRIX, Solaris, ...						

그림 9- 17. LSF의 층식구조

LSF의 구조

LSF구조의 층식표현을 그림 9-17에서 보여 준다. LSF는 대부분의 Unix가동환경들을 지원하며 JMS통신에 표준IP를 사용한다. 이것으로 하여 LSF는 Unix콤퓨터들의 이질적인 망을 하나의 클라스터로 전환할수 있다. 밑에 놓여 있는 OS핵심부는 변화시킬 필요가 없다. 다시 말하여 LSF는 가동환경독립이다.

말단사용자는 유틸리티지령들을 통하여 LSF의 기능들을 리용한다.

PVM은 지원되며 MPI지원은 예정되고 있다.

지령행대면부와 GUI가 제공된다.

LSF는 또한 숙련된 사용자들에게 LSLIB(load sharing library)라고 부르는 시간서고인 API를 제공한다.

LSLIB사용은 사용자가 응용코드를 수정하도록 명백하게 요구하며 따라서 유틸리티 지령들은 사용하지 않는다.

클라스터의 매개 봉사기마디에서 2개의 LSF데몬이 시작되여야 한다.

부하정보관리자(LIM)는 주기적으로 부하정보를 수집하여 교환한다.

원격실행봉사기(RES)는 임의의 과제들에 대하여 투명한 원격실행을 제공한다.

LSF유틸리티

LSF문헌에서는 콤퓨터를 호스트라고 부른다. 이 책의 나머지부분에서는 호스트대신에 일관하게 node(마디)라는 술어를 사용하겠다. 하나의 마디는 여러개의 처리기를 가질수 있지만 항상 조작체계의 단일한 복사만을 실행한다.

- LSF는 호상작용과 묶음, 순차, 병렬일감들의 모든 4개결합들을 지원한다. LSF를 통하여 실행되지 않는 일감을 **외래일감**이라고 한다. 봉사기마디는 LSF일감들을 실행할수 있는 마디이다. 의뢰기마디는 LSF일감들을 초기화하거나 제출할수 있지만 실행은 할수 없는 마디이다. 봉사기마디의 자원들만이 공유될수 있다. 봉사기마디들은 또한 LSF일감들을 초기화하거나 제출할수 있다.
- LSF는 LSF로부터 정보를 얻고 일감들을 원격으로 실행하기 위한 도구들의 모임(lstools)을 제공한다. 실례로 lshosts는 클라스터의 모든 봉사기마디의 정적자원들(아래를 보라.)을 기록한다. 지령 lsrun은 원격마디의 프로그람을 실행한다.

사용자가 의뢰기마디에 지령행 %lsrun−R ‘swp＞100’ myjob를 건반입력하면 응용은 100MB이상의 적용가능한 교환공간을 가지며 가장 가벼운 부하를 가지는 봉사기에서 자동적으로 실행된다.

- lsbatch유틸리티는 사용자들이 LSF를 통하여 묶음일감들을 제출하고 감시하며 실행하게 한다.
- lstcsh유틸리티는 공공Unix지령해석기 tcsh의 부하공유판본이다. 일단 사용자가 lstcsh대면부에 들어 가면 제출된 모든 지령은 국부마디를 포함하는 가장 적당한 마디에서 자동적으로 실행된다. 이것은 투명하게 진행된다. 즉 사용자는 국부마디에서 실행하는 tcsh와 꼭 같은 대면부를 본다.
- lsmake유틸리티는 Unix의 make유틸리티의 병렬판본이며 하나의 makefile이 여러 마디들에서 동시에 처리되도록 한다.

실례 9.14. 콤퓨터들의 클라스터에서 LSF의 응용

한 클라스터가 8개의 값 비싼 봉사기와 100개의 값 눅은 마디들(워크스테이션 또는 PC들)로 구성된다고 하자. 봉사기마디들은 응용쏘프트웨어를 포함하여 더 좋은 하드웨어와 쏘프트웨어를 사용하므로 비싸다.

하나의 사용허가는 Fortran 콤파일러와 CAD모의쏘프트웨어에 리용가능하며 4명까지

의 사용자에게 타당하다.

LSF와 같은 JMS의 사용과 봉사기마디들의 모든 하드웨어 및 쏘프트웨어자원들은 의뢰자들에게 투명한 방법으로 리용가능하다.

의뢰기의 말단앞에 앉아 있는 사용자는 마치도 국부적으로 의뢰기마디가 모든 쏘프트웨어와 봉사기의 속도를 가지는것처럼 느낀다.

《lsmake my.makefile》을 건반입력하여 사용자는 4개까지의 봉사기들에서 자기의 원천코드들을 콤파일할수 있다. LSF는 가장 적은 부하를 가지는 마디들을 선택한다. LSF의 사용은 자원리용에도 유리하다. 실례로 CAD모의를 실행하려는 사용자는 묶음일감을 제출할수 있다.

LSF는 CAD사용허가의 리용가능성을 자동적으로 검사하고 그것이 적용가능하게 되자마자 그 일감의 일정을 짠다. 사용자는 기다리지 않아도 된다.

많은 체계들에서 의뢰기는 대체로 휴식하고 있거나 일부 국부일감들을 실행하고 있다는데 주의해야 한다.

자원정보

매 봉사기마디의 LIM은 이 마디에 대한 정적 및 동적자원정보들을 수집한다. 정적자원정보는 시간에 따라 변하지 않으며 체계시동시에 LIM에 의해 얻어 진다. 동적자원정보는 부하벡토르의 부하지표들의 모임으로 표현되며 시간에 따라 변하고 LIM에 의해 주기적으로 얻어 진다.

- **정적자원** LSM이 공개한 일부 정적자원들을 표 9-5에 기입한다. 마디형태는 클라스터의 구성에 의해 결정되며 LSF관리자에 의해 설정된다. 그것은 Alpha, RS6000, MIPS 등과 같은 하나의 문자렬이다. CPU배수는 클라스터에서 가장 느린 마디 즉 CPU배수가 1인 마디와 비교한 마디의 상대속도이다.

다음 3개는 마디의 최대기억기공간과 국부디스크의 최대교환공간, 마디의 tmp파일체계의 최대공간이다.

표 9-5 LSF의 정적자원들

지표	척도	단위	결정요인
type	마디형태	문자렬	구성
cpuf	CPU배	상태	구성
maxmem	사용자에게 리용가능한 최대 RAM	MB	LIM
maxswp	사용자에게 리용가능한 최대교환공간	MB	LIM
maxtmp	tmp에서 리용가능한 최대공간	MB	LIM

다른 정적자원들은 쏘프트웨어의 사용허가와 리용가능성이다. 여기서 마디는 하나의 파일봉사기 등이다.

- **부하지표** 부하지표들의 일부를 표 9-6에 기입한다. 이 지표들은 매 1s 또는 30, 120s마다 LIM에 의해 갱신된다.

표 9-6 LSF 부하벡토르에 포함된 일부 부하지표들

지표	량	단위	평균	갱신간격
status	마디상태	문자렬	N/A	15s
R15s	실행대기렬길이	프로쎄스수	15s	15s
ut	CPU리용률	퍼센트	1min	15s
pg	페지속도	(페지수)per sec	1min	15s
1s	접속수	사용자수	N/A	30s
it	휴수시간	분	N/A	30s
swp	리용가능한 교환공간	MB	N/A	15s
mem	리용가능한 기억기	MB	N/A	15s
tmp	tmp의 리용 가능한 공간	MB	N/A	120s

마디의 상태는 ok(LSF일감수행에 리용가능), 비리용가능(LIM의 응답하지 않는다.), busy(부하가 턱값을 초과), 어떤 다른 상태일수 있다.

rl5s실행대기렬길이는 15s동안 CPU를 사용하기 위하여 준비된 처리의 평균수이다. CPU리용률은 지나간 시간에 CPU가 사용자 또는 체계코드들을 실행하고 있은 평균백분률이다.

- **페지화속도** 페지속도는 가상기억기페지속도 즉 매초당 디스크로부터 읽거나 디스크에 쓰는 페지의 수이다. 조밀하게 페지화되고있는 마디는 화상트랜잭션일감들에 느리게 응답한다.

1s지표는 이 마디에 접속한 사용자의 수로 잰다. 휴식시간은 건반누르기 또는 마우스누르기, 인쇄와 같은 마지막 입력 또는 출력트랜잭션으로부터 몇분이 흘렀는가를 잰다.

LSF는 아래에서 서술하는바와 같은 일부 부하공유책략들과 유용성지원, 부하평형전략들을 개발하였다.

부하공유방책

zhou et al.[668]은 여러가지 부하공유 및 일감배치방책들을 평가하였으며 LSF를 위한 다음의 방책들을 선택한다. 클라스터의 크기가 작을 때(실례로 수십개이하) LIM들중의 하나가 지배LIM으로 선출되며 나머지는 종속 LIM들이다. 종속 LIM들은 주기적으로 자기들의 부하벡토르를 지배LIM에 보내며 지배LIM은 그것들을 클라스터의 부하행렬에 결

합한다. 어떤 마디가 LSF일감을 제출할 때 일감실행은 지배LIM에 의하여 결정된다.

지배LIM

다른 말로 지배LIM은 중앙관리자로서 모든 봉사기로부터 온 부하정보들을 유지하며 클라스터의 모든 일감들을 처리한다. 더 큰 클라스터들(실례로 수백개의 봉사기마디들)에 대하여 이 단일지배책략은 적당치 않다.

LSF는 큰 클라스터를 몇개의 더 작은 부분클라스터들로 나눈다. 하나의 부분클라스터내에는 여전히 하나의 단일한 지배LIM이 있지만 지배LIM들은 교환하면서 부분클라스터호상간의 부하공유결정들을 협조적으로 채택한다.

표 9-7에서 보여 주는바와 같이 부하공유부가처리는 낮다.

표 9-7 LSF에 의한 일감응답시간, 지배 LIM의 부가처리,망통신흐름

클라스터의 구성	요청된 마리의 수			지배LIMCPU 시간%	망트레픽 (KB/s)
	1	5	10		
하나의 15개 봉사기 클라스터	3.8ms	4.6ms	5.0ms	0.8	0.04
하나의 33개 마디 클라스터	4.5ms	7.7ms	8.4ms	0.15	0.09
하나의 60개 마디 클라스터	5.8ms	14.2ms	18.5ms	0.29	0.17
20개의 50개 봉사기 부분클라스터들	N/A			1.1	1.2

실례 9.15. LSF부가처리의 평가

실례로 60개의 봉사기마디들로 이루어 지는 클라스터에 대하여 10개까지의 마디들이 요청될 때 일감요청이 나간 때로부터 그 일감이 시작될 때까지 걸리는 시간은 수미리초이다.

지배LIM은 마디의 CPU시간의 0.29%만을 소비하며 소비된 망대역너비는 0.17KB/S뿐이다.

50개의 마디를 포함하는 부분클라스터 20개로 이루어 진 하나의 클라스터에 대하여 지배LIM들의 CPU부가처리는 1.1%뿐이며 소비된 망대역너비는 1.2kb/S이다.

묶음지원(Batch Support)

LSF묶음체계구조를 그림 9-18에서 보여 준다. LSF는 묶음일감들을 위한 광범한 자원들을 제공한다. LSF는 NQS와 통합가능하며 이것은 초고속콤퓨터들을 위한 대중적인 묶음대기렬체계이다. LSF는 묶음일감을 실행하는 몇개의 묶음봉사기마디들을 사용한다.

묶음봉사기모임은 클라스터의 전체 봉사기모임의 부분모임이다.

종속묶음데몬 sbatchd는 모든 묶음봉사기에서 실행한다.

클라스터는 지배LIM이 실행하고 있는 단일한 지배묶음데몬 mbatchd를 가진다. 이 지

배마디는 모든 묶음일감대기렬들을 유지한다.

모든 묶음일감요청들은 지배묶음데몬에 보내지며 지배묶음데몬은 실행을 위해 일정짜기와 일감들의 종속묶음봉사와 마디들에로의 급송을 맡아 한다.

묶음일감의 제출

LSF일감은

bsub−q night −n 10−I jobin −o jobout −e joberr −R "mem>20" myjob

와 같은 bsub지령에 의해 제출된다. 이것은 myjob를 night라고 부르는 대기렬에 제출한다. myjob의 입력, 출력, 오유파일들은 각각 jobin, jobout, joberr이다.

일감은 매 마디가 적어도 20MB의 리용가능한 기억기를 가지는 10개 마디우에서 실행된다.

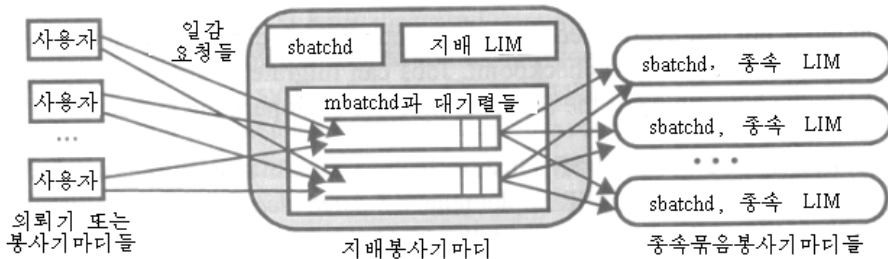



그림 9−18. LSF 묶음체계의 구조

LSF묶음일감의 생명주기는 그림 9−19에서 보여 주는 상태변화들을 거친다. 일감이 LSF묶음대기렬에 제출되면 그것은 대기(PEND)상태로 들어 간다. 일감이 모든 자원과 일정짜기조건들을 충족하는 마디로 급송되면 그것은 RUN상태로 된다.

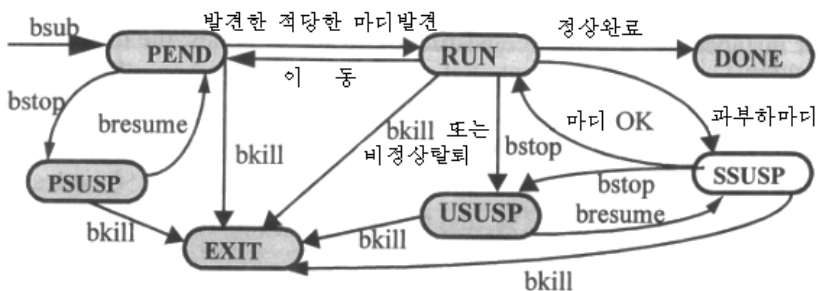



그림 9−19. LSF 묶음일감의 상태변화도

일감은 임의의 상태로부터 LSF의 bkill지령을 실행하면 소멸된다. 일감실행은 그것이 파손되면 비정상적으로 탈퇴한다.

일감은 일감의 소유자나 LSF관리자에 의하여 bstop지령을 통해 정지될수 있다. bstop

지령은 상태를 PEND로부터 PSUSP로 전환하며 또는 일감이 이미 급송되였으면 USUSP로 전환한다.

lsbatch체계는 또한 마디가 과부하이면 그 일감을 자동적으로 정지한다.

LSF는 먼저 온것은 먼저 봉사하기, 공정한 공유, 선점, 배제 등과 기본은 여러가지 일정짜기방법들을 지원한다.

공정한 공유방법은 자원을 자기의 몫보다 적게 소비한 일감부터 먼저 일정을 짠다.

선점일정짜기는 선점대기렬의 일감들이 마디의 더 낮은 우선권마디들 지어 그것들이 완성되지 않았다해도 선점하게 한다.

배제일감은 한 마디에서 홀로 실행한다. 일단 그것이 시작되면 lsbatch는 배제일감이 끝날 때까지 다른 LSF일감들이나 호상작용일감, 묶음일감들이 이 마디를 리용할수 없도록 자물쇠를 채운다.

유용성과 검사점설정

LSF는 유용성과 부하균형을 개선하기 위하여 여러 기술들을 통합한다. 결과 LSF 클라스터는 적어도 하나의 봉사기마디가 살아 있는 한 계속 동작할수 있다.

LSF는 대부분의 체계들에서 정적인 사용자수준의 서고를 통하여 묶음일감들에 대한 검사점설정을 지원한다.

핵심부수준의 검사점설정은 Convex OS가동환경용으로 실현된다.

묶음일감은 지배마디가 고장날 때조차도 잃어 지지 않는다. 일감들은 새 마디가 같은 구조와 같은 조작체계를 가지는 한 다른 마디로 동적으로 이동할수 있다.

LSF의 지배자선택

대부분의 LSF클라스터들은 단일고장점으로 되는 단일한 지배마디를 가진다. LSF는 이 문제를 잘 처리하기 위하여 지배자선발방법(그림 9-20)을 실현한다. LSF클라스터에서 LSM데몬들 LIM, RES와 종속묶음데몬 sbatch는 초기적재시에 모든 마디부에서 자동적으로 시작된다. 그림 9-20에서 보여 준것은 한개 마디의 상태변화이다.

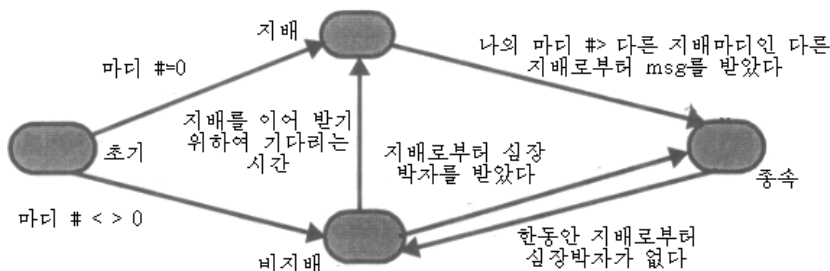



그림 9-20. LSF에서 지배자선발방법의 설명

임의의 LSF클라스터는 클라스터의 모든 봉사기마디들을 차례로 기록하는 구성파일을 유지한다.

매 마디는 하나의 마디번호를 가지는데 기록의 첫번째 마디의 번호는 0이다. 초기화후에 마디 0은 지배자로 된다. 지배자는 다른 봉사기마디들에 심장박자를 주기적으로 보낸다. 임의의 다른 마디는 먼저 기다리는 지배자로부터 하나의 심장박자를 받을때까지 비지배자상태에 들어 간다. 그다음 그것은 종속으로 된다. 지배자마디가 고장나면 심장박자통보문을 내보내지 않는다. 매 종속마다는 이것을 발견하고 비지배상태에 들어 간다.

그것은 마디번호에 비례하는 1개 시간주기동안 기다리고 새로운 지배자를 제출한다. 두개이상의 마디들이 경쟁하면 가장 작은 마디번호를 가지는 마디가 이긴다. 이 지배자선발방법은 클라스터에서 리용가능한 모든 봉사기마디들사이의 부하평형만큼 높은 유용성을 담보하도록 설계된다.

9. 6. 참고문헌주해와 련습문제

Gregory Pfister의 책[497]은 클라스터화개념에 대한 직관적인 소개를 제공한다. Martin et al.[431]은 클라스터들에서의 통신문제들을 론의하였다. Turcotte[620]는 클라스터콤퓨터를 위한 여러가지 쏘프트웨어도구들을 개괄한다. Walker et al.[633]은 클라스터들과 MPP들을 위한 열린 단일체계영상쏘프트웨어묶음을 개발하였다.

IBM HACMP클라스터는 Berstin[77]과 [338]에서 서술된다. Baker et al.[56]은 12개이상의 클라스터일감관리체계들을 개괄한다. 클라스터에 대한 일감일정짜기문제들은 Arpaci et al.[40], Feitelaon과 Rudolph[236], Saphire et al.[529], Du와 Zhang[213]에서 제기되였다. 부하평형과 고장허용은 Petri와 Laugendorfer[496], Rosenblum et al.[520], kumar et al.[384]에서 취급된다.

대표적인 콤퓨터체계들의 유용성자료(표 9-2)는 Digital[197]에 있다. 유용성체계들은 Gray와 Siewicrek[259], Bowen et al.[106], Azagury et al.[47]에서 연구된다. Sun Solaris MC는 khalidi et al.[369]에서 서술된다. LSF는 Zhou et al.[667, 668]에서 서술된다. Elnozhy et al.[228]과 Deconinck et al.[199]은 검사점설정과 복구회복기술에 대한 포괄적인 개괄을 주었다. 이 기술들은 Li et al, [415], Cao et al.[125], Plank et al.[503, 504, 505], Netz와 Xu[461], Wong과 Franklin[648]에서도 연구되였다. Tananenbaum과 Litzkaw[668]는 Condor체계를 론의하였다.

분산파일체계들은 Leuy와 Silberschartz[459], Santayarman[526], Vahalia[621]에서 개괄되였다. NFS veraion 3은 Pawlowski et al.[494], NFS의 Web판본은 Callaghan[120]에서 서술된다. AFS는 Morris et al.[453]에서 서술한 Andrew로부터 OSF[474]의 DCE 에서 서술한 DFS로 발전하였다.

문 제

문제 9.1. 클라스터와 관련된 다음의 술어들을 구별하고 실례를 드시오.

(1) 치밀한 클라스터 대 느슨한 클라스터
(2) 집중클라스터 대 분산클라스터
(3) 동질클라스터 대 이질클라스터
(4) 닫긴클라스터 대 로출클라스터
(5) 전용클라스터 대 기업클라스터

문제 9.2. LAN에서도 발견된 기업클라스터의 특징 4가지를 드시오. 그다음 클라스터가 보통의 LAN으로 되지 않게 하는 2가지 특징을 드시오. 공통적인 특징과 차이나는 특징들을 간단히 설명하시오.

문제 9.3. 9.2.1부분에서 서술되지 않았지만 체계의 예정되지 않는 고장을 일으킬수 있는 다른 상황들을 2개 드시오. 그러한 문제들을 해결하는 기술을 제기하시오.

문제 9.4. 그림 9-5 ㄴ)의 구성은 값이 비싼것으로 비판되고 있다.

(1) 2가지 다른 부족점을 들고 간단히 설명하시오.
(2) 그림 9-5 ㄱ)와 그림 9-5 ㄴ)사이의 절충으로서 자기자체의 높은 유용성체계를 설계하시오. 그리고 왜 자기의 설계가 더 좋은가를 설명하시오. LAN과 SDSI는 고도로 믿음직하며 이것들의 조종자도 믿음직하다고 가정한다. 또한 RAID디스크자체는 믿음직하지만 디스크조종자는 믿음직하지 않다고 가정한다.
(3) SCSI가 8개의 포구를 가질 때 설계를 7개 마디클라스터로 확장하시오.
(4) SCSI가 8개의 포구를 가질 때 설계를 16개 마디체계에 적당하게 수정하시오.

문제 9.5. 이 문제는 실례 9.3을 참고한다. 마디가 고장나면 고장을 진단하는데 10s, 작업부하가 교환되는데 또 30s 걸린다고 가정한다.

(1) 예정된 정지시간을 무시하면 클라스터의 유용성은 얼마인가?
(2) 클라스터가 정비를 위해 주당 1시간 정지하지만 한번에 한마디만 정지한다면 클라스터의 유용성은 얼마인가?

문제 9.6. Conder와 Linckpt를 비교하고 일반성과 유용성의 견지에서 각각의 우월성과 부족점을 론의하시오.

문제 9.7. 순차콤퓨터가 512MB의 주기억기와 충분한 디스크공간을 가진다고 하자. 큰 자료블로크에 대한 디스크읽기/쓰기대역너비는 1MB/S이다. 다음의 코드를 실행하고 검사점을 만들것을 요구한다.

```
do   1000  iterations
     A=foo(C  from   last   iteration)        /* 이 지령은 10분 걸린다. */
     B=goo(A)                                 /* 이 지령은 10분 걸린다. */
     C=hoo(B)                                 /* 이 지령은 10분 걸린다. */
end  do
```

A, B, C는 매개가 120MB의 배렬이다. 다른 모든것들(실례로 코드, 조작체계, 서고들)은 최대로 16MB의 기억기를 가진다. 콤퓨터가 정확히 한번은 고장나며 콤퓨터자체를 복구하는 시간은 무시된다.

(1) 검사점설정이 수행되면 최악의 경우 코드전체의 성공적인 완료에 대한 실행시간은 얼마인가
(2) 평범하고 투명한 검사점설정이 수행되면 최악의 경우 코드전체의 성공적인 완료에 대한 실행시간은 얼마인가?
(3) 포크식검사점설정을 (2)와 함께 사용하는것이 리로운가?
(4) 사용자지시검사점설정이 수행되면 최악의 경우 코드전체의 성공적인 완료에 대한 실행시간은 얼마인가? 최악의 경우의 실행시간을 최소화하기 위하여 사용자의 지령들이 첨가된 코드를 보이시오.
(5) 포크식검사점설정이 (4)와 함께 사용되면 최악의 경우 코드전체의 성공적인 완료에 대한 실행시간은 얼마인가

문제 9.8. 단일입력자료를 실현하는 실천적인 방법을 주고 그것이 어떻게 원천등록부문제와 다중접속문제를 해결하는가를 설명하시오. 방법은 ftp, telnet, http와 같은 대중적인 규약들을 지원해야 한다.

문제 9.9. 단일체계영상에 대한 다음의 술어들을 설명하고 실례를 드시오.

(1) 단일파일체계층
(2) 단일조종점
(3) 단일기억기공간
(4) 단일프로쎄스공간
(5) 단일I/O와 망

문제 9.10. Solaris MC를 고찰하고 다음의 물음에 대답하시오.

(1) 문제 9.9에서 어느 단일체계영상의 특징들이 Solaris MC에서 지원되는가? 지원되지 않는것은 어느것들인가?

(2) Solaris MC가 지원하는 그러한 특징들에 대하여 어떻게 지원하는가를 설명하시오.

문제 9.11. 클라스터일감관리체계들에 관한 다음의 술어들을 실례를 들어 설명하고 대비하시오.

(1) 순차일감 대 병렬일감

(2) 묶음일감 대 호상작용일감

(3) 클라스터일감 대 국부일감

(4) 클라스터의 프로세스, 국부프로세스 대 핵심부의 프로세스

(5) 전용방식, 공간공유방식, 시간공유방식

(6) 독립일정짜기 대 무리일정짜기

문제 9.12. 이 련습들은 LSF에 대한것이다.

(1) LSF일감의 4개 형태 매개의 실례를 드시오.

(2) 외래일감의 실례를 드시오.

(3) 1000개 봉사기클라스터에 대하여 왜 LSF부하공유방법이 전체 클라스터가 1개의 지배LIM을 가지는것보다 또는 모든 LIM들이 지배자인것보다 더 좋은가 하는 2가지 리유를 드시오.

(4) LSF지배자선발방법에서 비지배자상태의 마디가 새로운 지배자를 제출하기전에 마디번호에 비례하는 시간주기동안 기다린다. 왜 기다림시간이 마디번호에 비례하는가?

제 10 장. 봉사기들과 워크스테이션들의 클라스터

최근 년간에 콤퓨터클라스터제품들과 업무와 기업들, 연구기관들에서 그것들의 응용이 눈에 뜨이게 나타났다. 대부분의 클라스터들은 더 높은 유용성과 확장가능한 성능을 강조한다.

아래에서는 클라스터콤퓨터분야를 고찰하고 높은급으로부터 낮은 가격의 체계들까지의 클라스터들에 대한 지원경향을 개괄한다. 여러 클라스터체계들 즉 Microsoft Wolfpack, SGI POWER CHALLENGEarray, Digital TruCluster, Berkeley NOW프로젝트, Rice Treadp Mark DSM클라스터에 대하여 상세히 연구한다. 이러한 연구는 오늘날의 클라스터제품들과 클라스터연구프로젝트들에 대한 연구 및 응용의 경험들을 보여 준다.

Wolfpack는 Windows NT봉사기클라스터의 클라스터화에 대한 판매자교차(cross-vendor)쏘프트웨어표준을 개발하는 공업프로젝트이다. SP는 과학 및 업무응용들을 위하여 성공적으로 시장에 내놓은 상업클라스터 MPP를 나타낸다. TruCluater는 기억기통로기술에 영향을 주는 SMP봉사에로의 클라스터이다. NOW는 Unix워크스테이션들의 클라스터화를 위한 일부 새로운 기구들을 탐구하기 위한 대학연구과제이다. TreadMarks는 쏘프트웨어 실현된 워크스테이션들의 DSM클라스터이다.

비교를 위하여 그밖의 클라스터체계들과 과제들을 이 장에서 간단히 소개한다.

10. 1. 클라스터제품들과 연구과제

Pfister[498]에 의하여 평가된바와 같이 세계적으로 100000개이상의 콤퓨터클라스터들이 사용되고 있다. 이 클라스터들에서 마디들은 대체로 PC들과 워크스테이션들, SMP봉사기들이다. 클라스터크기는 대체로 수십개 마디정도이며 몇개의 클라스터들만이 100개 마디를 넘는다.

대부분의 클라스터들은 그들사이의 정규적인 LAN접속을 제외하고 마디들을 접속하는데 고속 또는 기가비트이써네트, FDDI고리들, ATM 또는 Myrinet교환기들과 같은 상품적망들을 사용한다. 일부는 IBM SP2과 Compaq NT클라스터와 같은 전용의 높은 대역너비의 망들을 사용한다.

일부 공공령역 및 상업쏘프트웨어묶음들은 단일체계영상과 더 높은 유용성, 분산일감관리를 지원하는데 리용가능하다.

기본적인 클라스터화기술들이 존재한다. 업무응용들에서 클라스터제품에 대한 대량생산을 시작할 기회는 급속히 다가오고 있다.

10. 1. 1. 클라스터제품들에 대한 지원동향

콤퓨터들에 대한 클라스터화지원을 높은급시장으로부터 고볼륨시장으로 이동하고 있다.

그림 10.1에서 묘사된바와 같이 클라스터화지원은 체계복합체를 위한 IBM

Sypiex[339]와 SGI POWER CHALLENGE array[548]과 같은 큰 대형콤퓨터(mainframe computer)들을 한데 련결하는것으로부터 시작하였다.

높은급콤퓨터의 목적은 중요한 기업인터네트와 자료채취응용들에서 협조적인 무리계산과 더 높은 유용성에 대한 요구를 쉽게 실현하는것이다.

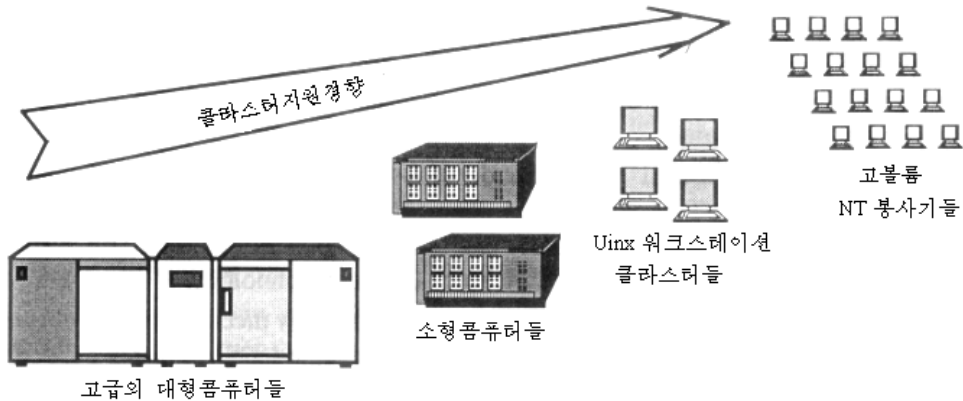



그림 10-1. 높은급대형콤퓨터로부터 Unix워크스테이션 및 고볼륨 PC체계에로의 클라스터화지원경향

점차적으로 그 경향은 여러 AXP들과 VAX들이 같은 디스크/테프조종기들의 모임에 련결된 Digital의 Open VMS클라스터[551]에서와 같이 여러 소형콤퓨터들에 대한 지원으로 움직이고 있다.

Tandem의 Himalaya[605]는 고장허용의 직결트랜잭션처리(OLTP)를 위한 고급클라스터를 설계하였다.

탁상용 또는 책상면워크스테이션사용에 대한 늘어 나는 경향과 함께 Unix워크스테이션들의 클라스터화는 주요워크스테이션판매자들속에서 대단히 대중화되였다.

일부 상업클라스터체계들을 표 10-1에 기록한다. 이 표는 모든 판매자들의 클라스터제품들을 남김없이 포함하는것은 아니다. 표는 국부적으로 조립된 많은 클라스터들을 포함하지 않는다. 아래에서는 이러한 이미 조립된 상업클라스터들을 간단히 소개한다. 보다 상세한 정보는 WWW홈페지나 오른쪽 렬에 기입된 참고서로부터 찾을수 있다.

좋은 워크스테이션클라스터의 실례에는 Sun SPARC클라스터[599]와 HP의 Apollo 9000콤퓨터클라스터, Sequent의 Symmetry 5000, Digital의 Alpha워크스테이션봉사소[194] 등이 있다.

최근동향은 Intel형PC들의 클라스터화에로의 이동이다. 클라스터제품들은 통합체계, 통합쏘프트웨어도구, 통합된 유용성하부구조, 통합조작체계확장 등으로 나타난다. 클라스터화동향은 콤퓨터산업에서의 크기감소동향과 조화되여 큰 시렁(rack)크기의 고급콤퓨터체계로부터 대용량의 탁상형콤퓨터체계로 이동하고 있다.

더 작은 마디들의 클라스터지원은 많은 클라스터대변자들이 예측한대로 용량상에서의 판매를 늘일것이다. 이 동향은 Intel형PC들과 저가격워크스테이션들의 광범한 사용에 의하여 일어 나고 있다. IBM과 Digital로부터 Sun과 SGI, HP, Compaq, Microsoft까지의

전체 공업은 클라스터화시장으로 하여 흥분되고 있다. Intel의 SHV체계와 4통로 Pentium Pro에 기초한 주기, Windows NT의 대칭다중처리능력의 유용성은 저가격봉사클라스터들에 대한 시장능력을 크게 높이였다.

표 10-1 상업클라스터체계제품들에 대한 견본표

회사	체계이름	간단한 서술과 참고서들
DEC	VMS-Clusters	VMS를 위한 고유용성클라스터들 [55]
	TruClusters	SMP봉사기들의 Unix클라스터[127],[397]
	NT Clusters	WindowsNT를 위한 Alpha식클라스터[195]
HP	Apollo 9000 Cluster	계산클라스터[497]
	MC/ServiceGuard	NT클라스터 Solution을 위한 HP NetServer[573]
IBM	Sysplex	상업적묶음일감과 OLTP를 위한 공유디스크대형콤퓨터 클라스터[339]
	HACMP	고유용성클라스터다중처리기[338]
	Scalable POWERparallel (SP)	POWER 2마디들과 확장기준HPP로서의 Omega 교환기로 구성된 워크스테이션클라스터[14]
Microsoft	Wolfpack	Windows NT봉사기들의 클라스터화를 위한 열린 표준[446]
SGI	POWER CHALLENGEarray	분산식병행을 위하여 HiPPI교환기로 구성한 SMP봉사기마디들의 확장가능클라스터[548]
Sun	Solaris MC	Solans Sun워크스테이션클라스터의 확장[369]
	SPARCcluster 1000/2000 PDB	OLTP와 자료기지처리를 위한 고유용성 클라스터봉사기[599]
Tandem	Himalaya	OLTP와 자료기지를 위한 2중마디들을 가지는 고도로 확장가능한 결패허용의 클라스터[605]
Matathon	MIAL2	완전한 여유와 실패복구를 가지는 고유용성클라스터[425]

클라스터화는 유용성과 성능을 다 높일수 있다.

표 10-11에서 보여 주는바와 같이 대부분의 체계들은 유용성클라스터로 구성된다. 다른것들은 확장가능한 성능을 강조하는 콤퓨터클라스터로서 사용된다. 클라스터의 2가지 목표는 반드시 모순되는것은 아니다. 일부 고유용성클라스터들은 Tandem의 Himalaya체계실례와 같이 확대가능성을 위하여 진취적으로 하드웨어여유를 사용한다. 대부분의 유용성클라스터들은 클라스터의 하드웨어여유를 개발하기 위하여 이미 구성된 쏘프트웨어기술들을 사용한다. 고성능클라스터들은 흔히 고성능의 망과 망대면부, 통신쏘프트웨어를 포함하며 특별히 설계된 통신부분체계에 의해 지원된다. 2개의 실례는 기억기통로기술을 리용하는 DEC TruCluster와 고성능교환기를 리용하는 IBM SP2이다.

10. 1. 2. SMP봉사기들의 클라스터

명백한 공업적경향은 여러개의 동질SMP봉사기들을 하나의 통합된 초고속봉사기(supersewer)로 클라스터화하는것이다. 8.2에서 제기된 Sun의 Ultra Emterprise 10000초고속봉사기는 좋은 실례로 된다. 또 다른 초고속봉사기실례는 아래에서 소개하는 SGI의 POWER CHALLENGEarray이다.

실례 10.1. SGI의 SMP클라스터 POWER CHALLENGEarray

SGI의 초고속봉사기클라스터의 대략적인 블로크도를 그림 10-2에서 보여 준다. 이 체계는 초고속봉사기로서의 클라스터로 형성하기 위하여 POWERnode라고 하는 2～8개의 SMD마디들을 련결한다. 매 POWERnode는 36개까지의 MIPS 10000처리기와 16GB의 공유기억기를 가지는 Power Challenge SMP봉사기이다.

총체적으로 초고속봉사기클라스터는 128GB까지의 주기억기, 4GB/S이상의 확장된 디스크전송능력, 28TB의 디스크용량을 가질수 있다. RAID기억용량은 139.2TB에 이를수 있다. 마디들은 높은 대역너비의 통신을 지원하기 위하여 하나의 크로스바 HiPPI교환기로 접속된다. 2등분대역너비들은 최대구성에서 1.6GB/S로 평가된다. 체계는 Indy워크스테이션들로부터 이써네트를 통하여 접근될수 있다.

체계관리자 Console은 단일조종점이다. 이것은 ST-1600box를 통하여 매 마디에로의 직접적인 RS232접속을 가진다(점선들).

이 클라스터는 매 POWERnode의 공유기억기를 가지고 동작한다. 통보문넘기기는 POWERnode들사이에서 충분히 지원된다.

Silicon Graphics클라스터는 2～288개의 MIPS Ⓡ R1000Ⓡ 처리기들로 이루어 진다. Silicon Graphics초고속봉사기의 주목할만한 새로운 하드웨어 및 쏘프트웨어특징들을 아래에 개괄한다.

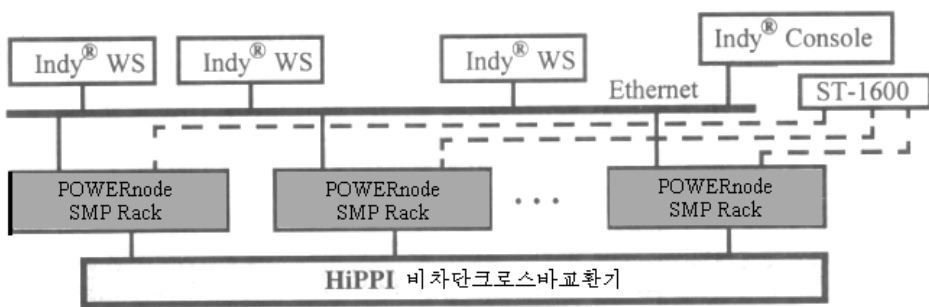



그림 10-2. SMP 봉사기들의 Silicon Graphics클라스터

- SMP봉사기들의 클라스터는 분산병렬처리에로의 모듈식방법을 준다. 충분히 리용될 때 최대성능은 109Gflop/S이다.
- 그것은 공유기억기와 통보문넘기기기술을 결합한다. 이것은 전통적인 SMP봉사기들과의 쏘프트웨어호환성을 제공한다.

- SMP마디들사이의 고대역너비 HiPPI크로스바접속으로 하여 비정규평형에 대하여 통신 대 통신비가 감소된다.
- 모든 POWERnode들은 64b Unix® OS를 개선한 Silicon Graphics® 6.1을 실행하며 호상작용적인 초고속계산에 대한 고성능그라픽스들을 제공한다.
- 매 POWERnode의 공유기억기접근지연은 1μs 이하이며 크로스바교환기접속시간은 1μs 이하이다.
- 프로그람작성은 모듈들의 계층 즉 Fortran과 HPF, C콤파일러들을 병렬성하고 공유기억기프로그람작성, PVM과 MPI지원을 가지는 통보문넘기기를 사용한다.

10.1.3. 클라스터에 대한 연구과제

클라스터화는 활발한 연구분야이다.

고속통신, 일감일정짜기, 단일클라스터과제들을 표 10-2에 제시한다. 역시 이 표는 진행중에 있는 모든 과제들을 포함할수 없다. 여기서 보여 주는것은 고도로 새로운 발전이 이룩되고 있는 일부 실례과제들이다.

아래에서는 이 클라스터과제들을 간단히 특징 짓는다. 기록된 클라스터과제들 매개는 일부 가치 있는 독특한 특징들을 개발하였다. 특징들은 단일체계영상의 쏘프트웨어지원과 같은 구조적인 혁신과 유용성, 인터네트와 인트라네트지원들을 가지는 대규모분산성능평가기준응용들을 포함한다.

표 10-2 대표적인 클라스터연구과제들

과제 /참고서	클라스터화를 지원하는 특별한 특징들
Berkeley NOW 과제[33]	능동통보문과 협조적인 파일, 대역적인 Unix확장으로 특징 지어 지는 워크스테이션들의 봉사기 없는 망(10.5)
Princeton SHRIMP[237]	상품적구성요소들과 특별한 망대면부를 통한 효률적인 통신과 공유가상기억기
Karsruhe Parastation[636]	분산병렬처리를 위한 효률적인 통신망과 쏘프트웨어개발
Rice TreadMarks[30]	워크스테이션들의 쏘프트웨어실현, 분산공유기억기, 클라스터
Wisconin Wind Tunnel[516]	상품적망으로 호상접속된 워크스테이션들의 클라스터에서의 분산공유기억기
NSCP[133] http://nscp.upenn.edu	인터네트로 련결된 3개의 국부클라스터우에서 메타콤퓨터를 위한 확장가능클라스터과제
Argonne Globus[200]	북아메리카의 ATM접속된 17개 싸이트들의 WAN을 통한 메타콤퓨터가동환경과 쏘프트웨어개발
Syracuse WWVM[207]	인터네트와 HPCC기술을 가지는 고성능계산을 위한 World-Wide 가상기계
PearlCluster[328] http://pearl.cs.hku.hk	분산다매체와 금융수자서고응용들을 위한 연구클라스터 (실례6.10)
Virginia Legion[282]	민족가상콤퓨터기지를 지향하는 메타계산쏘프트웨어개발

- NOW과제는 쏘프트웨어지원, 단일체계영상, I/O와 파일체계, 효률적인 통신, 개선된 유용성을 포함하는 클라스터콤퓨터화문제의 거의 전체 범위를 포괄한다. 10.5에서 NOW과제[33]를 상세히 연구한다.
- Princeton SHRIMP과제[237]는 값 비싼 PC형체계들로 만들어 진 클라스터에서 효률적인 통신과 공유기억기의 지원에 초점을 둔다. 특별한 망대면부기판들이 페지이동수준에서 공유가상기억기를 달성하기 위하여 개발된다.
- Winconsin Wind Tunnel과제에서는 상품적마디들과 상품적망으로 구성된 느슨하게 결합된 클라스터에서 어떻게 캐쉬일관성공유기억기를 실현하는가를 연구하고 있다.
- Rice대학의 TreadMarks[30]은 워크스테이션들의 쏘프트웨어실현공유기억기클라스터의 좋은 실례이다. 기억기공유는 사용자공간의 실시간서고와 함께 실현된다. 10.6에서 TreadMarks를 상세하게 고찰한다.
- NSCP(National Scalable Cluster Project)는 메타콤퓨터체계들을 연구한다[133]. 이것은 지리적으로 먼 지역들에 있는 여러 도시들에 뻗어 있는 이질클라스터들이다. NSCP견본체계는 시카고와 메릴랜드, 펜실바니아의 3개의 대학클라스터들로 구성되여 인터네트를 통하여 접속된다. 상세한 내용은 Web싸이트 http://nscp,vrenn,edu를 참고하시오.
- 류사한 메타콤퓨터화과제에는 Globus과제가 있는데 이것은 북아메리카의 17개 싸이트들우에서 초고속콤퓨터들과 대용량기억체계들, 시각화장치들을 접속하는 ATM망우에 실현된다[200, 248, 249].
- Legion과제는 민족가상콤퓨터기지를 위한 메타콤퓨터화쏘프트웨어[282,410]를 개발하고 있다.
- WWVM(World−Wide Virtual Machine)과제는 인터네트와 HPCC(High Performance Computing and Communication)기술을 목표로 하고 있다(고성능콤퓨터실현을 목표로 하고 있다.)[207, 252].
- 홍콩대학의 Pearl클라스터는 실례 6.10에서 소개하였다. 이것은 Sun과 HP, Alpha, SGI의 Unix에 기초한 봉사기와 워크스테이션들을 ATM접속한 클라스터이다. 고유한 특징에는 SSI를 위한 middleware지원과 두 나라 말을 리용하는 인터네트검색엔진, 쟈바와 련결된 MPI통신부분체계가 있다. 이 특징들은 금융수자서고와 분산다매체응용들을 위하여 개발된다.

10. 2. NT클라스터를 위한 Microsoft Wolfpack

이 책이 완성되였을 때 Microsoft는 Windows NT들을 클라스터화하기 위한 판매자교차(cross−vendor)표준의 첫 판본을 내놓았다. 처음에 그 과제는 Microsoft외에 Intel과 Compaq, Dihital, HP, NCR, Tandem의 핵심성원들이 망라되였다. 전략적인 협조로 Pfister가 도입한 이름 Dogpack로부터 착상된 코드이름 Wolfpack를 가지는 새로운 클라스터표준을 만들었다.

１０．２．１．Microsoft Wolfpack의 구성

Wolfpack과제는 Windows NT봉사기들을 위한 클라스터쏘프트웨어의 열린 표준을 개발하기 위하여 1995년 후에 시작되였다. 그것은 또한 표준적인 클라스터화쏘프트웨어의 이름일수도 있다. Wolfpack는 다음의 독특한 보증을 가진다.

- Wolfpack는 사용자들과 체계판매자들, 3부류쏘프트웨어개발자들에게 응용프로그람작성대면부들에 대한 열린 명세서들을 제공하는 열린 표준이다. 이것은 확정적으로 응용쏘프트웨어와 체계쏘프트웨어, 클라스터하드웨어의 개발을 촉진할것이다.
- Wolfpack는 상품적PC나 Intel형봉사기가동환경들, SCSI기억모선들과 표준망들에 잘 맞는다. 이것은 개발시간을 줄이고 가격의 효과성을 제공하기 위한것이다. 그러나 낮은 지연의 망과 같은 고성능하드웨어의 사용은 포함하지 않는다.
- Wolfpack의 장기적인 목표는 PC들과 워크스테이션들, 고볼륨봉사기들의 클라스터가 NT조작체계에서 실행하든 비NT조작체계에서 실행하든 클라스터의 유용성뿐아니라 확대가능성, 리용편리성을 강화하는것이다.

Wolfpack의 구성요소들

Wolfpack는 아래에서 간단히 소개하는바와 같은 클라스터의 API들, NT클라스터지원, 클라스터 solution의 모임을 서술한다.

(1) **Wolfpack API** API표준과 SDK(Software Development Kit)는 응용클라스터들을 이식하도록 설계된다. API접근은 고장회복을 촉진하며 고장사건들에 대하여 사용자에게 통지하고 비표준지원들을 재획득하며 단순파손이나 잠금을 제외한 복잡한 고장을 감시할수 있다. Wolfpack API는 또한 NT클라스터에서 확대가능성과 동적부하균형을 촉진할수 있다.

(2) **NT클라스터지원** 이것은 NT봉사기응용이 Wolfpack에 복종하도록 한다. 이것은 Wolfpack가 임의의 NT봉사기응용에 대한 기본적인 고장회복을 진행한다는것을 의미한다. 포장(wrapper)동적련결서고(DLL)는 클라스터관리자에게 통지하고 기본적인 심장박자신호들을 창조하기 위하여 개발되였다. 그것은 오늘 Windows NT가 대칭적인 다중처리지원을 가지는것과 같은 방법으로 Microsoft가 클라스터지원을 NT봉사기들에 첨가하려는것처럼 보인다.

(3) **클라스터와 Solution** Wolfpack는 여러 단계로 개발된다. 16개 마디클라스터에 대한 지원은 1998년 중순에 Beta검사에 들어 간다. Microsoft는 Wolfpack를Intel형체계들이외에 Alpha와 POWerPC, MIPS형봉사기들에도 리용가능하게 만들것을 위임 받고 있다. 또한 NT봉사기구조는 32통로까지의 SMP연산들을 지원한다. Wolfpack는 앞으로 32개 또는 그이상의 마디들로 확대되도록 될것이다.

유용성클라스터구성

초기의 Wolfpack클라스터 solution은 2개의 봉사기마디들에 대하여 3개의 높아 지는

유용성수준으로 유용성지원을 제공한다. 그것들은 9.2.2에서 소개하는것과 같은 즉시대응, 활동이어 받기, 고장허용이다.

3가지 Wolfpack클라스터구성들을 회복시간과 고장복구능력마디의 활동성의 견지에서 표 10-3에 서술한다. 유용성의 수준은 대기클라스터구성으로부터 활동 및 고장허용클라스터구성으로 가면서 높아 진다.

회복시간이 짧을수록 클라스터의 유용성은 높아 진다.

고장복구는 수리 혹은 정비후에 정상동작으로 돌아 오는 회복된 마디의 능력이다.

활동성은 정상동작기간에 그 마디가 능동적인 작업에 사용되는가 하는것이다.

이 클라스터들은 다음의 부분들에서 론의된다.

표 10-3 Wolfpack가 지원하는 3가지 유용성클라스터구성

구성	회복시간	고장 복구능력	마디의 활동성	다중봉사기클라스터 판매자들
즉시대응	40-200s	아니	대기마디에 대하여 아니	Vinca, IBM, Octopus, Compaq
활동이어 받기	15-90s	예	예	Microsoft, Digital Compaq, Tandem, NCR,HP, Amdahl, Stratus
고장허용	~1s	예	예/아니	Marathon

10. 2. 2. 즉시대응다중봉사기클라스터

즉시대응다중봉사기클라스터에서는 보통 주마디만이 유용한 작업을 활동적으로 진행하고 있다. 대기마디는 전원이 투입되면 기본마디의 상태를 검사하기 위한 심장박자신호들을 통신하는 일부 감시프로그람들을 실행하고 있으며 다른 유용한 작업부하는 활동적으로 실행하지 않고 있다.

주마디는 임의의 자료를 공유디스크기억에 반사해야 하는데 이것은 대기마디에 의해 접근하기 쉽다.

대기마디는 모든 자료에 대한 충분한 두번째 복사를 요구한다.

그림 10-3에서 대표적인 대기클라스터구조를 보여 준다. 2중모선은 단일모선의 단일

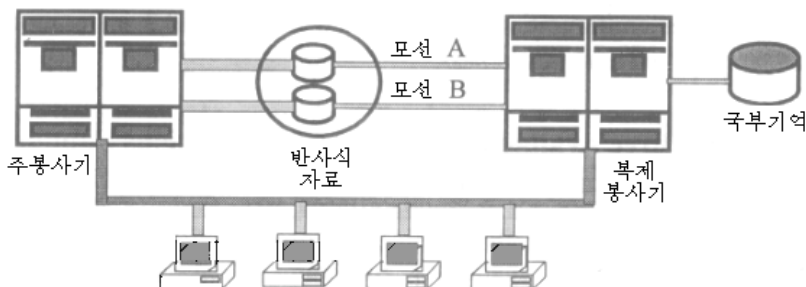



그림 10-3. 즉시대응다중봉사기클라스터의 구성

고장점을 제거하는 2개의 봉사기들에 접속하며 클라스터에 더 높은 유용성을 준다. 주마디가 고장나면 거기에서 실행하는 일감은 대기마디를 걸쳐서 고장회피(failover)될수 있으며 이것은 낡은 마디가 수리되는 동안 새로운 주마디로 된다. 그러나 대부분의 대기클라스터들에서 고장회피는 가역이 아니다.

대기클라스터화를 지원하는 판매자들을 표 10-4에서 개괄한다. 대기클라스터는 제한된 고장회복능력으로 하여 가격이 더 낮다.

이 대기클라스터들중 일부는 사용자들을 Windows NT와 Windows 95, Windows for Workgroups(WFW)3.11로부터 새로운 활동봉사기에로 자동적으로 절환할수 있다.

표 10-4 공업용대기봉사기클라스터들

클라스터판매자	클라스터모형	봉사기/디스크 가동환경	제공하는 클라스터지원
Vinca	Vanica대기봉사기	Intel에 기초한 봉사기들과 SCSI조종자들	Netware와 OS/2, NT를 지원
IBM	PC봉사기 고유용성 sdution	Vinca의 대기봉사기가동환경을 가지는 무리 IBM PC 봉사기하드웨어	OS/2, NT. Notes클라스터, PB2클라스터
Octopus Technologies	ASO를 통한 자동절환	Intel과 Alpha봉사기들	NT와N통로 Failover
Compaq	대기회복봉사기와 직결회복봉사기	Compaq봉사기들, 외부기억함, SCSI기판들, 하드웨어 호상접속카드	쏘프트웨어보다는 전용하드웨어를 통한 failover

10. 2. 3. 능동적인 유용성클라스터

그림 10-4에서는 활동이어 받기(계승)구성을 가지는 유용성클라스터의 구조를 보여준다.

이 경우 구조는 2개의 봉사기마디사이에서 대칭적이다. 두 봉사기는 정상적으로 유용한 작업을 수행하는 주마디이다. 고장회피(failover)와 실패복구(failback)는 흔히 두봉사기마디들에서 지원된다. 한 마디가 고장나면 사용자의 응용들은 클라스터의 리용가능한 마디에로 고장회피한다.

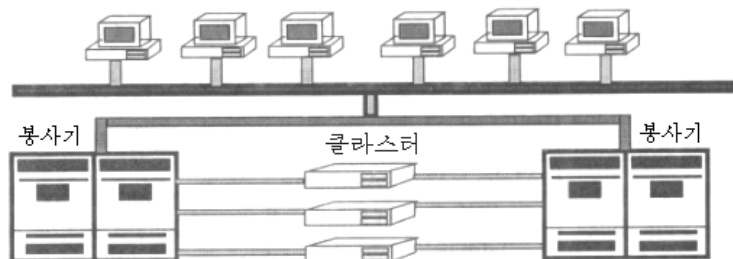



그림 10-4. 2중능동클라스터의 구조

표 10-5에 기입된 8개의 판매자들은 활동클라스터를 구성하기 위한 쏘프트웨어 및 하드웨어를 공급한다. Microsoft와 Digital의 제품들은 각각 주로 Intel형NT봉사기들과 Alpha형봉사기들을 지원한다. 나머지판매자들은 일단 NT클라스터화묶음이 자기들의 하드웨어망 플채트홈으로 키를 돌리면 Wolfpack추종을 지원할것이다.

사용자들은 고장회피의 실현에 요구되는 시간에 따라 일부 지연을 느끼거나 마지막 검사점에 보존하지 않았던 일부 자료를 잃게 될것이다.

수동적인 부하평형이 고장회피조건들로 하여 발생한 보충적인 부하를 최소화할수 있거나 하나의 활동클라스터의 두개이상의 마디들에 대한 과부하상황을 늦출수 있다. 자동고장복구의 특징은 기초적인 대기클라스터들에 비한 주요개선을 가져 온다.

Tandem은 NT클라스터 solution을 가지지 않지만 Tandem의 Compaq에로의 통합은 활동NT클라스터나 고장허용Tandem클라스터를 생각하는데서 공동의 전문가적인 견해를 강화한다. Himalaya클라스터들은 OLTP와 전자상 거래, 인터네트WWW, 자료창고, 결심지원, 소매, 건강관리, 수송시장을 포함하는 1000개이상의 중요한 업무응용들을 지원한다.

Wolfpack특징들중의 일부는 비NT봉사기마디들을 가지는 클라스터들을 지원하는데로 확장되고 있다. NCR Lifekeeper와 HP MC/Service Guad, Amdahl의 En Vista Servers, Stratus RADIO는 그러한 능동적인 클라스터의 실례들이다. 고장회피와 고장복구능력은 이 모든 봉사기클라스터들에 존재한다. 특히 RADIO는 2개의 콤퓨터봉사기와 2개의 기억장치, 2개의 망이 하드웨어와 쏘프트웨어, 망의 모든 단일고장점들을 제거하는데 사용되는 자체의 충분한 여분에 대하여 특히 강하다.

10. 2. 4. 고장허용다중봉사기클라스터

이 구조는 유용성클라스터에 대한 가장 높은 수준의 개발을 가리킨다.

고장허용은 충분한 여유와 능동적인 이어 받기능력에 의해 달성된다. 모든 단일고장점들은 고장허용능력을 제공하기 위하여 복제된다. CPU들과 조작체계들, 기억기모선들, 망대면부들, 디스크구동장치들, 전원공급, 체계망 등과 같은 모든 중요한 구성요소들이 모두 복제된다.

Tandem과 Marathon와 같은 몇개의 콤퓨터회사들만이 고장허용클라스터들을 제품으로 내보냈다. Tandem Himalaya는 높은급사용자들을 위하여 판매되였다. Marathon클라스터는 아래에 실례로 준다.

표 10-5　　능동클라스터판매자들과 그 제공품들

판매자	클라스터제품들 또는 기술	마디/망 가동환경	클라스터화지원과 wolfpack추종
Microsoft	Wolfpack	Intel에 기초한 Alpha(현재)와 Power PC에 기초한 봉사기(미래)	Wolfpack에서 windows NT, API, SDK, DLL지원
Digital	NT클라스터, NT클라스터+Pack	Alpha NTFS, Microsoft SQL Server 6.5, Oracle 7/8 봉사기들	Intel 및 Alpha형봉사기들에 대한 고장회피 지원

표계속

판매자	클라스터제품들 또는 기술	마디/망 가동환경	클라스터화지원과 wolfpack추종
Compaq	직결회복봉사기	Compaq SCSI교환과 Tandem ServerNet	비자동오유복구, wolfpack호환, ServerNet호상접속
Tandem	NonStop Serverware 봉사기들, ISV Portfdio, ServerNet	Compaq NT 봉사기들, SMP 봉사기들의 Himalaya클라스터들	Compaq에로의 wolfpack, SQL자료기지, TUXEDO, CICS API들, OLTP, 업무응용들
NCR	여러 봉사기들을 위한 회복도구를 가지는 Lifekeeper	Intel형봉사기들, oracle7병렬봉사기, TCP/IP, NetBEDI, SQL봉사기들	Windows NT, 자동오유복구와 자동재접속을 가지는 wolfpack지원
HP	MC/Service Guard, HP Wolfpack	HP 9000 Unix, NT봉사기를 실행하는 NETServer, Oracle병렬봉사기	Wolfpack에서 NT클라스터 solution을 지원
Amdahl	다중봉사기 클라스터확장	Intel SHV형En Vista봉사기들, 고속이써네트, 40MB/s Fujitsu 호상접속	En Vista유연성관리자, 고장회피, wolfpack를 지원
Stratus	RADIO클라스터	2개의 Pentium에 기초한 마디, 2개의 PCI고속폭, SCSI-2디스크구동장치, 2개의 100BaseT집선기	하드웨어, 쏘프트웨어,망에서 임의의 단일고장을 극복하는 모든 능동클라스터특징들을 지원하는 Isis유용성관리자

실례 10.2. Marathon고장허용다중봉사기클라스터

그림 10-5에서 off-the-shelf구성요소들로 구성된 고장허용클라스터 Marathon MIAL2의 구조를 보여 준다. 클라스터요구는 능동적인 모든 클라스터부분과 다른 구성요소들에 대한 여유를 가지는것이다. 클라스터의 고장회피시간은 1s내에서 유지되여야 한다.

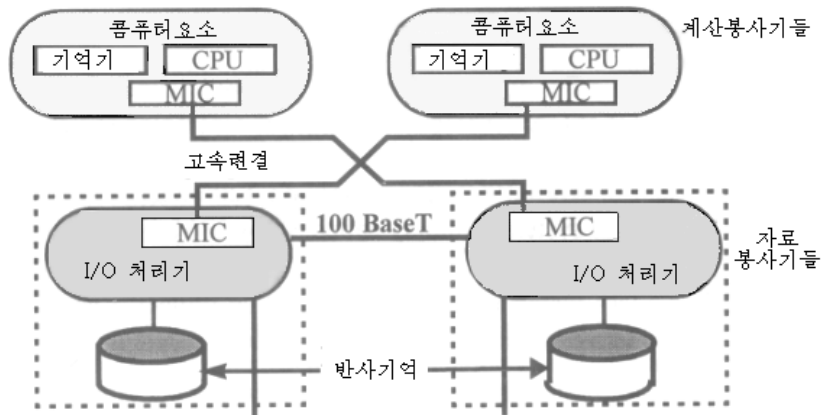



그림 10-5. Marathon MIAL2 고장허용클라스터

목표는 한해에 6분이하의 정지시간을 가지는 99.999%의 유용성을 달성하는것이다. 이것은 대단히 높은 요구이며 흔히 좀 비싸다.

MIAL2는 최신CPU소편들로 구성된 2개의 콤퓨터봉사기를 가진다. 2개의 자료봉사기들은 완전중복고속련결들과 MIC(Marathon interface card)로 2개의 콤퓨터봉사기에 접속된다. 콤퓨터봉사기는 실시간자료보호를 제공하기 위하여 정보를 2개의 자료봉사기에 동시에 쓴다. 전용 100Base T련결들은 반사식기억디스크들을 가지는 2개의 자료봉사기들속에 존재한다.

만일 하나의 자료봉사기가 고장나면 사용자의 응용은 두번째 자료봉사기에서 계속된다. 이것은 하드웨어에 의한 무결성검사와 32MB/S처리량에 의해 달성된다.

고장난 봉사기가 수리되면 리용가능한 자료봉사기는 전체 디스크기억을 회복된 봉사기로 사본함으로써 회복된 봉사기를 자동적으로 재동기화한다. 이 사본은 반사식기억령역 1GB당 1min의 속도로 진행된다. 복제와 활동성으로 MIAL은 모든 형태의 단일고장들을 허용할수 있다.

Marathon은 자기의 클라스터가 API들과 스크립트들, NT의 특정한 판본을 요구하지 않는다고 주장하고 있다. 그러나 Marathon은 자기의 클라스터가 Wolfpack확장클라스터에 하나의 마디로 참가하기를 바라고 있다.

10. 3. IBM SP체계

IBM SP의 특징은 클라스터방법을 MPP구성에 사용한다는데 있다. 그 설계경험은 잘 문서화되여 있으며 대규모고성능체계들을 설계하는데서 많은 식견을 준다. IBM의 설계목표들과 전략들에 대한 개괄을 준 다음 SP클라스터의 구조와 쏘프트웨어환경을 론의한다.

10. 3. 1. 설계목표와 전략

IBM은 1991년 가을 확장가능한 POWERparallel(SP)과제[646]를 시작하면서 MPP업무에 들어 갈것을 결정하였다. 개발집단은 1992년 2월에 무어 졌으며 18개월미만에 첫번째 제품을 내보냈다. 사실 SP1이라고 부르는 첫번째 제품은 1993년 4월에 사용자들에게 배포되였다. follow-on 12개 마디 SP2체계들은 1994년 8월에 배포되였다.

지금은 체계를 간단히 SP라고 부른다. 1998년까지 전 세계에 3000개이상의 SP체계들이 설치되였다. SP는 IBM Deep Blue과제의 하나인 세계서양장기경기에 적용되였다.

아래에 서술하는것은 클라스터같기도 하고 MPP같기도 한 SP체계들의 성공에 영향을 주는 요인들이다.

- **시장출하시간** 고성능콤퓨터는 Moore의 법칙에 류사한 경향을 따른다. 선도적인 가격 및 성능을 얻자면 제품이 짧은 시간에 개발되는것이 불가피하다.
- **일반목적** SP는 서로 다른 기술적 및 상업적응용들과 류행되는 프로그람작성모형들, 서로 다른 동작방식들을 지원하는 일반목적체계여야 한다.
- **고성능** SP는 최대성능은 아니라도 높은 확대된 성능을 제공해야 한다. 이것은

많은 고속처리기들뿐아니라 고속기억기, 고속통신, 좋은 콤파일러, 서고 등에도 의존된다.

- **유용성** SP는 고객들이 자기의 상업제품들을 실행할수 있도록 훌륭한 믿음성과 유용성을 제공해야 한다.

이 목표들을 만족시키기 위하여 IBM집단은 융통성 있는 클라스터구조 즉 전용호상접속, 개선된 고성능봉사들과 유용성지원들, 표준프로그람작성모형들, 선택적인 단일체계영상을 지원한다는 전략을 채택하였다.

클라스터구조

IBM집단이 내린 가장 힘들고 중요한 결정은 클라스터구조를 선택하는것이였다. 그 기본특징들은 다음과 같다.

- 매 마디는 자체의 국부디스크를 가지는 RS/6000워크스테이션이다.
- 완전한 AIX(IBM의 Unix)가 매 마디에 상주한다.
- 마디들은 고속망을 통하여 접속된다. 망대면부들은 성길게 결합된다. 즉 마디는 국부기억기모선을 통해서가 아니라 자기의 I/O모선을 통하여 망에 접속된다.

IBM은 이러한 구조가 2가지 주요한 우월성 즉 단순성과 융통성을 가진다고 보고 있다. 이 구조는 구성하기 간단하며 서로 다른 응용요구들을 만족하도록 구성할수 있다. 그것은 IBM에서 이미 개발한 Vulcan분사기억기기술과 RS/6000워크스테이션기술을 사용할수 있다(Vulcan은 IBM 내부의 다중콤퓨터연구견본의 이름이다.).

SP계렬은 가능한껏 표준적인 워크스테이션구성요소들을 사용한다. 전용쏘프트웨어와 하드웨어는 표준기술이 확장가능체계에 대한 성능요구를 만족할수 없을 때에만 개발된다. 클라스터구조선택은 SP가 4개의 설계목표 특히 시장출하시간과 일반목적목표를 달성하도록 하는 열쇠로 된다.

융통성 있는 구조

융통성 있는 구조를 가지는것은 SP가 서로 다른 구성들을 허락한다는것을 의미한다. 체계는 몇개의 마디로부터 수백개의 마디로 크기가 확장가능해야 한다. 마디들은 응용과 환경에 대한 사용자의 요구들을 만족시키도록 개별적으로 (하드웨어적으로 또는 쏘프트웨어적으로) 구성가능해야 한다.

전용호상접속

IBM은 빛섬유통로와 ATM과 같은 현존 상품적망들이 MPP에 요구되는 통신지연과 대역너비를 제공하는데 적당치 않다고 보고 있다. 그래서 IBM은 Vulcan과제에서 개발된 다단망기술을 사용하기로 결정하였다. 다단망기술은 확장가능SP체계들을 구성하는데 사용되는 고성능교환기(HPS)호상접속으로 되고 있다.

표준환경

SP는 표준적인 열린 분산 Unix환경을 사용한다. 이것은 MPP가 기계특정의 OS쏘프트웨어개발의 힘든 과제를 피할수 있게 한다. SP는 워크스테이션클라스터이므로 분산 Unix환경에서 리용가능한 체계관리와 일감관리, 기억관리, 자료기지, 통보문넘기기를 위한 현존 표준쏘프트웨어를 사용할수 있다. 이 모든 쏘프트웨어는 이미 IBM워크스테이션의 AIX조작체계에 존재한다.

고성능봉사들

전통적인 분산AIX환경에서 효률적으로 실행될수 없는 현존 또는 새로운 응용들을 위하여 SP는 고성능봉사를 제공한다. 고성능봉사에는 다음의것들이 포함된다.

- 고속호상접속망(HPS)
- 효률적인 사용자수준통신망규약(US규약)
- 최량화된 통보문넘기기서고(MPL)
- 병렬프로그람개발과 실행환경
- 병렬파일체계
- 병렬자료기지(실례로 병렬 DB2)
- 고성능I/0본문체계

표준프로그람작성모형

일반목적의 목표를 달성하기 위하여 IBM은 3가지 령역에서 류행되고 있는 프로그람작성모형들을 지원하기로 결정하였다.

- **순차계산** SP는 병렬콤퓨터이지만 현존의 순차프로그람들을 단일마디에서 변화없이 실행하게 하여야 한다. 클라스터의 구조와 표준환경때문에 SP2에서 이 과제는 쉽게 달성된다. 전통적인 C와 C++, Fortran의 제공을 제외하고 SP는 RS/6000워크스테이션들을 위하여 개발한 10000개이상의 응용들을 그 어떤 수정도 요구하지 않고 실행할수 있게 한다.
- **병렬기술계산** SP는 현재 통보문넘기기 및 자료병렬(data−parallel)프로그람작성(HPF)모형들을 지원한다. 공유기억모형에 대한 지원은 계획되고 있다.
- **병렬상업계산** 상업응용들을 지원하기 위하여 IBM은 몇개의 기본자료기지와 트랜잭션감시부분체계들을 병렬성하고 있다. IBM DB2자료기지체계의 병렬판본(DB2병렬판 또는 DB2 PE라고 부른다.)이 SP2에서 실현되였다.

체계유용성

SP체계는 수천개의 상품적부분품들로 구성된다. 이러한 하드웨어 및 쏘프트웨어구성요소들은 본래 값 눅은 워크스테이션들을 위하여 개발된것이지 큰 고장허용체계들을 위하여 개발된것이 아니다. 그것들이 유용성량들을 가지지 않으면 자주 실패하여 전체 체계를 파피시킨다. SP는 체계유용성을 높이기 위하여 다음의 기술들을 채택하였다.

- 클라스터구조는 매 마디에 하나씩 분리된 조작체계형태를 포함한다. 한 형태의 고장은 전체 체계를 무능하게 하지 말아야 한다. 이것은 SMP구조에 비하여 우월하다. SMP구조에서 단일조작체계형태는 공유기억기에 상주하며 OS의 고장은 전체 체계를 무능하게 만든다.
- SP설계는 전체 체계를 무능하게 만드는 단일고장점들을 체계적으로 제거한다. 실례로 마디들은 2개의 망 HPS와 이써네트에 의하여 접속된다. HPS가 고장나면 마디들은 이써네트를 통하여 통신한다.
- 쏘프트웨어구조는 고장발견과 고장진단, 체계재구성, 고장회복을 위한 봉사들을 제공한다. 이 하부구조는 SP가 깨끗하게 퇴화되도록 한다.

선택식단일체계영상

분산체계에서 사용자는 개별적이고 분리된 워크스테이션들을 만난다. 진짜 단일형태를 가지는 체계에서 사용자는 거대한 워크스테이션(실례로 하나의 조작체계형태)을 만난다.

IBM은 서로 다른 사용자 및 환경의 요구들을 조사하였으며 진짜 단일체계는 실현하기 힘들고 일부 상업응용들에서 결정적인 요구가 아니라는것을 발견하였다. 결과 IBM은 2개의 극단사이의 타협을 결정하였다. SP체계는 단일입력자료, 단일파일계층, 단일조종점, 단일일감관리체계의 SSI특징들을 실현하였다.그밖의 요구되는 SSI특징들중에서 단일주소공간은 SP체계에서 실현되지 않는다. 이것은 SP구조와 함께 전진하면서 명백하게 된다.

１０．３．２．SP2체계의 구성방식

SP의 간단화된 론리구조를 그림 10-6에서 보여 준다. SP는 2～512개의 마디들로 구성되며 매 마디는 자체의 국부기억기와 국부디스크를 가진다. 매 SP는 또한 체계console처럼 봉사하는 하나의 R/6000워크스테이션을 요구한다.

체계호상접속

마디들은 2개의 망 즉 전통적인 이써네트와 HPS에 의해 접속된다. 이써네트는 느리지만 다음과 같은 리익을 준다.

- 이써네트는 통신성능이 중요치 않은 프로그람개발에 사용될수 있다. 개발된 코드는 제품실행을 위하여 HPS를 사용할수 있다.
- 이써네트는 HPS가 고장나면 복제로 사용된다. 이써네트가 없으면 HPS는 단일고장점이다.
- 이써네트는 동시조종의 우월성을 가지므로 SP개발시간을 줄이게 한다. HPS와 관련된 쏘프트웨어가 개발과 개선중에 있는 동안 이써네트는 체계의 나머지 부분이 개발되고 결함수정되며 검사되고 사용되도록 한다.
- 이써네트는 또한 체계감시와 초기기동, 적재, 검사, 다른 체계관리에 사용될수 있다.

6장에서 이미 HPS구조(실례 6.5)와 크로스바교환기설계(실례 6.4)를 고찰하였으므로 여기서는 상세한 내용을 반복하지 않는다.

마디구조

그림 10−6에서 보여 주는바와 같이 매 마디는 하나의 사설(private)기억기와 국부디스크를 가진다. 이써네트와 HPS에로의 접속은 느슨하게 결합된다. 즉 그것들은 I/O모선을 통하여 접속된다. 이것은 NIC가 기억기모선에 치밀하게 결합된 Intel Paragon이나 Cray T3D와 대조된다.

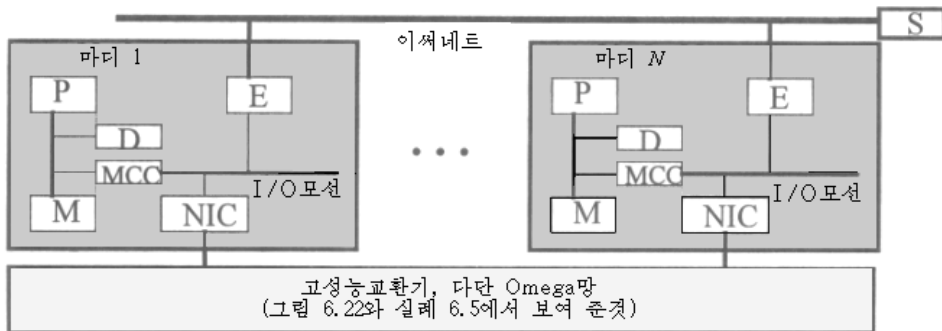



그림 10−6. SP체계구조(P:처리기, M:기억기, D:디스크, MCC:MicroChannel 조종기, NIC: 망대면부교환기, E:이써네트적응기, S:체계 console)

SP는 구성의 융통성을 효과적으로 지원하는 3가지 물리마디형태 즉 넓은 마디와 좁은 마디, 좁은 마디 2를 제공한다(실례 1.1을 참고). 이 3가지 마디형태들은 그림 10−7에서 비교된다.

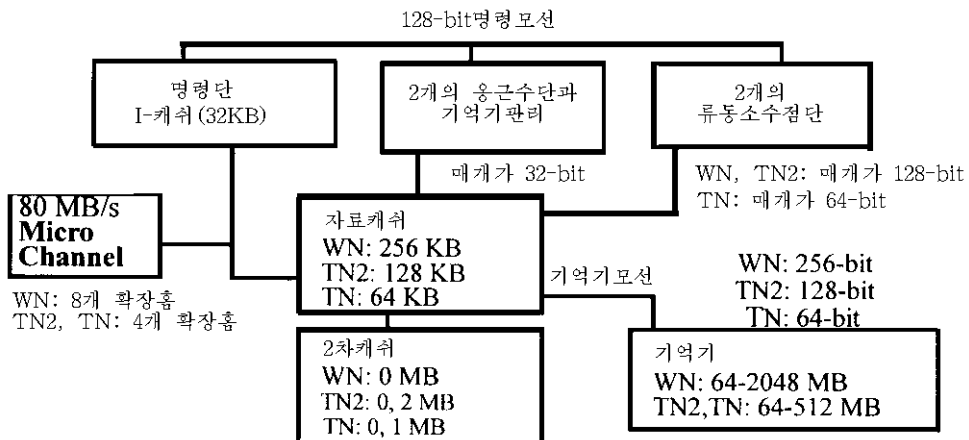



그림 10−7. SP체계의 넓은 마디(WN), 좁은 마디 2(TN2), 좁은 마디(TN)에서 처리기마디의 구조

모든 마디들은 박자가 66.7MHz인 하나의 POWER2 다중처리기를 사용한다. 매 처리

기는 32KB의 명령캐쉬와 자료캐쉬, 명령 및 분기조종단, 2개의 정수단, 2개의 류동소수점단을 가진다. 여기서 매단은 한 박자주기에 하나의 곱하기와 하나의 더하기를 실행할수 있다. 이것은 POWER2처리기에 4×66.7=267Mflop/S의 첨수속도를 준다. POWER2는 매 박자주기당 6개의 명령을 수행할수 있는 초고속스칼라처리기이다. 짧은 명령관흐름들과 정교한 분기예측기술들, 등록기기술들로 하여 POWER2은 하나의 박자주기에 2개의 적재/기억, 2개의 류동소수점곱하기와 더하기, 하나의 지수증가, 하나의 조건분기를 수행할수 있다.

3가지의 마디형태들은 계층기억기의 용량과 자료경로폭, I/0모선확장홈들에서 차이난다.

실례로 하나의 넓은 마디는 2GB까지의 주기억기와 256KB의 자료캐쉬, MicroChannel우의 8개의 I/0확장홈들을 가진다. 기억기모선은 256b폭이며 2.1GB/S의 최대대역너비를 제공한다. 4통로련합자료캐쉬는 자료를 주기당 4개의 64b연산자의 최대속도로 2개의 류동소수점단을 공급할수 있다.

좁은 마디들은 더 작은 최대 기억기/캐쉬용량을 가지며 4개까지의 I/0확장홈을 허용한다.

하나의 좁은 마디 2는 1MB(2MB)의 2차캐쉬를 가질수 있다. 큰 기억기/캐쉬용량과 대역너비, 초고속스칼라설계, 좁은 콤파일러는 SP가 매 마디에 확대된 성능을 줄수 있게 한다. 이것은 SP의 리용이 NAS성능평가기준을 계산하는데서 왜 다른 MPP들보다 더 좋은가를 부분적으로 설명한다.

10. 3. 3. I/0와 망호상접속

SP체계에 대한 I/0부분체계와 망대면부들, 망호상접속요구들을 아래에 준다.

SP I/0부분체계

SP I/0부분체계의 구조를 그림 10-8에서 보여 준다. I/0부분체계는 본질적으로 SP체계밖의 다른 기계들에 LAN관문과 함께 HPS주위에 구성된다.

SP마디들은 4개의 부류로 구성될수 있다. 호스트마디(H)는 호상작용적인 프로쎄스와 마찬가지로 여러가지 사용자가입대화를 처리하는데 사용된다. I/0마디들은 주로 대역파일

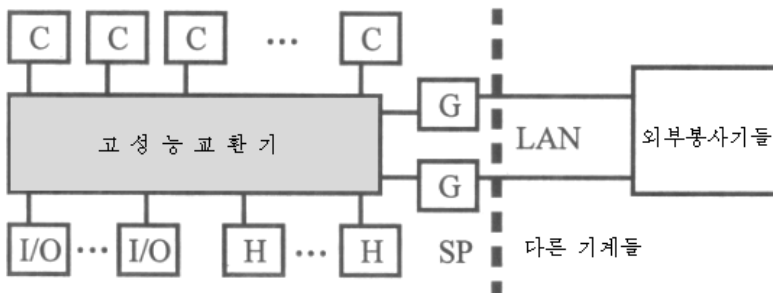



그림 10-8. SP I/0부분체계(C:콤퓨터마디, G : 관문마디, H : 호스트마디, I/0 : 표준 또는 병렬 I/0봉사기마디, LAN : 이써네트, 통표고리, FDDI, H : PPI, ATM)

봉사기들과 같은 I/0기능들을 수행한다. 관문에서(G)는 망화기능들을 봉사한다. 계산마디(C)들은 계산에 리용된다. 이 4개의 부류는 겹칠수 있다. 실례로 호스트마디는 계산마디, I/0마디, 관문마디일수 있다.

외부봉사기들은 파일봉사기들과 망경로조종기들, 시각화체계들과 같은 외부의 기계들을 SP에 추가한다.

망대면부

매 SP마디는 그림 10-9에서 보여 주는바와 같이 교환기적응기 또는 통신적응기라고 하는 망대면부회로를 통하여 MPS에 접속된다. 적응기는 8MB의 DRAM 한개를 포함하며 40MHz의 Intel i860극소형처리기에 의해 조종된다. 적응기는 MicroChannel에 련결된다. MicroChannel은 주변장치들을 RS/6000들과 IBM PC에 접속하는데 사용된다.

매 적응기는 기억기 및 절환엔진리단(MSMU)이라고 하는 소편을 거쳐서 HPS에 련결된다. MSMU에서 들어 오는 FIFO와 나가는 FIFO에 접속되는 쌍방향련결은 매개가 8b폭을 가지는 2개의 통로로 구성된다. 2개의 FIFO외에 2개의 조종상태등록기를 포함하는데 이것은 DRAM을 검사하고 재생하는 i860모선조종기로도 봉사한다. BIDI라고 불리우는 4KB의 쌍방향FIFO(매 방향에 2KB)는 MicroChannel과 i860모선을 접속한다.

적응기는 큰 국부통보문완충기를 요구하는 서로 다른 규약들을 수용하기 위하여 큰 (8MB) DRAM을 사용한다. 마디의 처리기는 프로그람식I/0지령들을 통하여 적응기의 DRAM과 MSMU에 직접 접속할수 있다. 그러나 DMA를 사용하는 자료전송은 BIDI를 통하여야 한다.

한 마디로부터 HPS에로의 자료전송은 다음과 같이 진행된다.

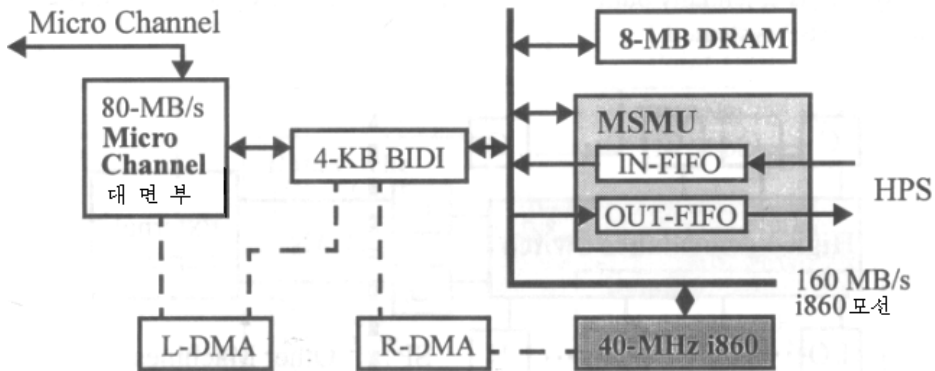



그림 10-9. SP통신적응기(MSMU : 기억기 및 절환엔진리단, BIDI : 완충기억기, DMA : 직접기억접근엔진)

마디의 처리기는 보낼 자료를 적응기에 알린다. i860은 BIDI의 머리부에 쓴다. BIDI의 머리부는 직접기억기접근(DMA)전송에 필요한 정보를 포함한다. 머리부가 BIDI의 앞에 도착하면 왼쪽 DMA엔진(L-DMA)이 넘겨 받아 그 자료를 그 마디(Micro Channel)로부터 BIDI로 전송한다. 전송이 끝나면 L-DMA는 하드웨어계수기를 증가시킨다. 그다음 i860은 또 다른 머리부를 오른쪽 DMA엔진(R-DMA)에 쓰며 R-DMA는 자료를 BIDI로부터

MSMU의 OUT−FIFO에 전송한다. 그다음 OUT−FIFO는 그 자료를 HPS에 전송한다.

자료수신도 비슷하다. 자료가 도착한 후 MSMU는 i860에 알리며 i860은 머리부에 오른쪽 DMA를 시작하도록 쓴다. R−DMA는 IN−FIFO로부터 BIDI에로의 전송을 이어 받는다. 전송이 끝난후 i860은 머리부를 BIDI에 쓴다. L−DMA는 그것이 BIDI의 앞에 도착하면 머리부를 추출하고 자료를 BIDI로부터 MicroChannel로 전송한다. 적응기의 구조는 동시적인 동작을 허용한다.

실례 10.3. MHPCC의 400개 마디 SP2다중콤퓨터

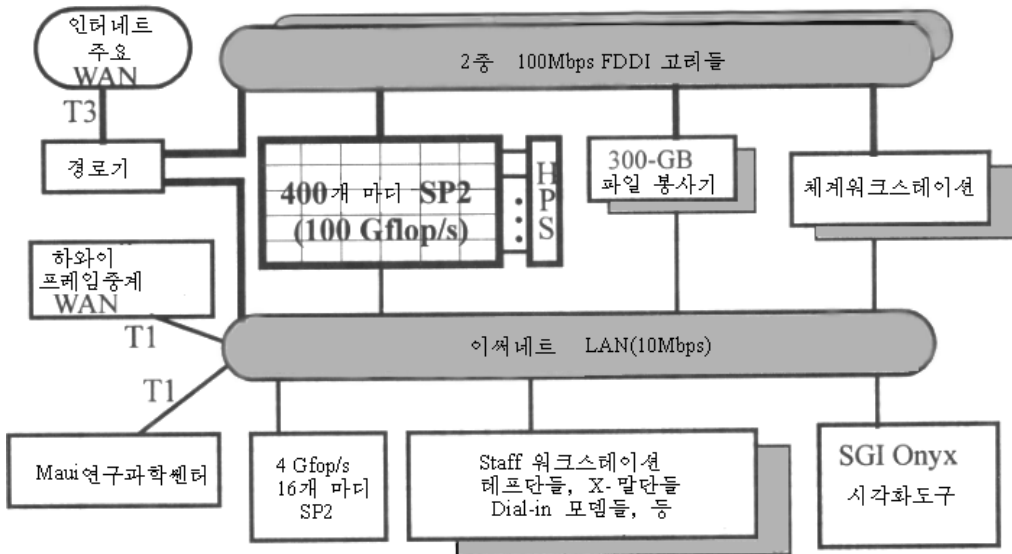
세계에서 가장 큰 SP2구성들중 하나를 그림 10−10에서 보여 준다.

HPS=고성능교환기, WAN=광지역망
FDDI=빛섬유분산자료대면부, T3선 (45 Mbps),
T1 (선들) (1.54 Mbps). 얇은 선들은 10-Mbps 이써네트접속들이다.

그림 10−10. Maui 고성능콤퓨터쎈터에 설치한 400 개 마디 IBM SP2 체계

이것은 1994년에 Maui고성능콤퓨터쎈터(MHPCC)에 설치한 400개 마디 클라스터이다. MHPCC는 하와이섬의 Maui섬에 위치하고 있다.

다른 더 큰 체계는 Conell리론쎈터의 512개 마디 SP체계이다. HPS이외의 모든 마디들 또는 2중FDDI고리와 10Mbps이써네트에 의해 접속된다. FDDI(100Mbps)고리들은 고유용성에 필요한 하드웨어여유를 제공한다. 병렬응용에서 통보문통신은 3개의 선택 즉 HPS, FDDI, 이써네트를 가진다. 체계관리와 유용성, 파일연산들, 망접근들과 같은 다른 모든 통신들은 FDDI나 이써네트망들을 통하여 진행할수 있다. 이 SP2체계는 많은 고속과학계산과 대규모단일처리응용들에 적용되였다.

10. 3. 4. SP체계쏘프트웨어

SP체계쏘프트웨어계층은 그림 10-11에서 보여 준다. 이것은 핵심부 IBM AIX조작체계이다. SP는 다음과 같은 대부분의 Rs/6000워크스테이션환경을 재사용한다.

- 10000개이상의 순차응용들
- 자료기지관리체계(실례로 DB2)
- 직렬트랜잭션처리모니터들(실례로 CICS/6000)
- 체계관리와 일감관리
- Fortran, C, C++콤파일러들
- 수학적 및 공학적서고들(실례로 FSSL)
- 표준 AIX조작체계

그림 10-11. SP체계쏘프트웨어계층(PE:처리환경)

SP체계는 어떤 새로운 쏘프트웨어를 첨가하고 확장가능병렬클라스터체계에 요구되는 일부 현존 쏘프트웨어를 개선하기만 한다.

병렬환경(PE)

AIX PE는 그림 10-12에서 보여 주는바와 같이 자기의 병렬프로그람을 개발하고 실행하는 사용자들을 위한 가동환경이다. 그것은 4개의 구성요소 즉 병렬조작환경(POE)과 통보문넘기기서고(MPL), 시각화도구(VT), 병렬결함수정기(pdbx)를 가진다.

병렬조작환경(POE)

POE는 병렬프로그람들의 실행을 조종하는데 사용된다. 그 구조를 그림 10-12에서 보여 준다. 실행은 SP 또는 SP마디에 접속된 RS/6000워크스테이션인 원천마디에서 실행하는 분할관리자의 프로쎄스에 의해 조종된다. 원천마디는 사용자가 병렬프로그람을 호출하는 곳인데 병렬프로그람은 SP의 콤퓨터마디의 하나 또는 그이상의 과제로서 실행한다.

원천마디는 표준Unix I/0장치들(실례로 stdin, stdout, stderr)을 제공한다. 그것은 LAN(실례로 하나의 이써네트)을 통하여 콤퓨터마디들과 표준 I/0통신을 수행한다. 실례

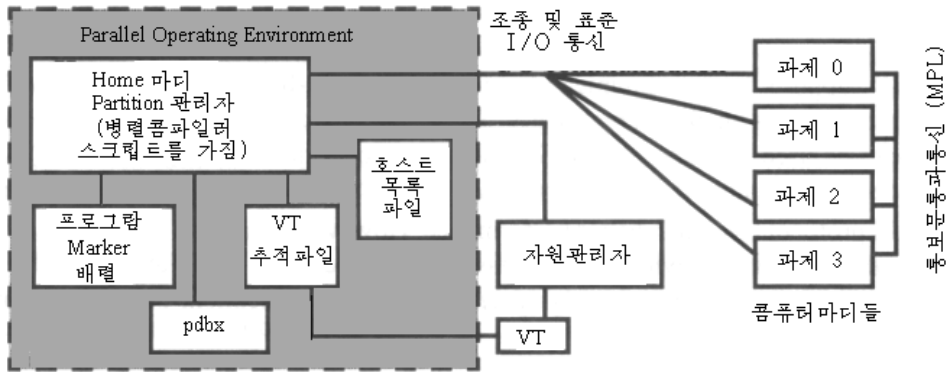



그림 10-12. IBM클라스터에 구성된 PE와 POE

로 사용자는 원천마디의 건반에서 Control-C를 눌러 모든 과제들을 끝낼수 있다. printf명령문에 의한 출력은 원천마디의 현시장치에 표시된다.

MPL과 MPI

통보문넘기기통신은 특정한 MPL함수들을 HPS 혹은 이써네트를 통해 실행함으로써 실현된다. 이 서고는 프로쎄스관리와 묶기(grouping), 점대점통신, 통신을 위한 총 33개의 함수를 제공한다. IBM SP는 또한 본래부터 실현된것을 포함하여 MPI의 서로 다른 판본들을 지원한다.

고성능봉사들

IBM SP의 클라스터구조는 본래 RS/6000워크스테이션들과 TCP/IP망에 기초한 분산체계들을 위하여 개발된 표준적인 off-the-shelf쏘프트웨어의 즉시적인 사용을 가능하게 한다. 그밖에 SP는 전통적인 분산체계환경에서는 효률적으로 실행할수 없는 새로운 응용들과 부분체계들을 위한 여러가지 고성능봉사를 제공한다. 이 봉사에는 고성능통신부분체계와 고성능파일체계, 병렬서고들, 병렬자료기지, 고성능I/0가 포함된다. 여기서는 이 봉사들중 몇개만을 론의한다.

SP는 2개의 통신규약 즉 핵심부공간에서 실행하는 IP에 기초한 규약(실례로 UDP/IP 또는 TCP/IP)과 US라고 부르는 사용자공간의 규약을 지원한다. 두 규약은 HPS나 전통적인 망(실례로 이써네트)에서 사용된다.

HPS에서 IP와 US규약들의 성능을 표 10-6에서 보여 준다. US규약은 더 좋은 성능을 가진다. 그러나 SP는 US규약을 사용하는 매 마디에 하나의 과제만을 허용한다. 한편 매 마디의 다중과제들은 HPS를 공유하는데 IP를 사용할수 있다.

표 10-6 SP에서 2개 규약의 성능

규약	지연(ms)	대역너비(MB/s)
UDP/IP	277	10.8
US	39	35.5

고성능통신을 요구하지 않는 응용들에 대하여 IP사용은 더 좋은 전체적인 체계리용을 가져 온다.

병렬 I/0파일체계

SP고성능파일체계를 PIOFS(Parallel I/0 File System)라고 부른다. 이것은 대부분의 응용들과 체계유틸리티들을 위한 POSIX-read와 write, open, close, ls, cp, mv와 같은 Unix연산들을 순차 Unix체계에서와 같이 동작시킨다. 그것은 AIX에서 2^{32}대신 2^{64} B까지의 큰 파일크기를 지원한다. 전통적인 Unix파일체계대면부를 따르는외에 PIOFS는 파일에서 병렬분산과 병렬연산을 가능하게 하는 병렬대면부를 제공한다.

IBM은 SP우에서 다른 클라스터가동환경에서만큼 실행하는 DB2병렬판이라고 부르는 병렬자료기쏘프트웨어프로그람을 개발하였다. 이것은 공유 없는 구조에 기초하며 함수이동기술을 사용한다. 자료기지는 여러 마디들에 분산되며 자료기지함수는 자료가 상주하는 마디로 이동된다. DB2병렬판은 기계크기와 문제크기에 맞게 확장가능하다. 이것은 수백개의 마디에서 실행할수 있으며 테라바이트의 자료를 포함하는 큰 자료기지를 다룰수 있다.

유용성봉사

SP체계는 마디들에서 실행하는 데몬들의 모임을 통하여 쏘프트웨어유용성하부구조를 제공한다. 심장박자데몬들은 어느 마디들이 살아 있는가를 가리키는 심장박자통보문들을 주기적으로 교환한다. 성원봉사마디들과 프로쎄스들이 일정한 무리의 부속물로써 식별되도록 한다.

마디고장과 닫기, 재시동에 의해 생긴 성원변화사건에서 통지봉사들은 활동성원들을 그 사건에 통지하는데 사용되며 다음에 활동성원들이 작업을 계속할수 있도록 회복절차들을 조화시키기 위하여 회복봉사기들이 호출된다.

대역봉사

여러 대역봉사들은 단일체계영상에 선택된 형태들을 제공한다. 외부의 체계자료저장소는 체계의 마디들과 교환기들, 현재의 일감들에 대한 체계정보들을 유지한다. SDR는 체계의 부분이 고장날 때 다른 부분들에 영향을 주지 않고 그 체계를 재구성하는데 쓸모있다. SDR의 내용들은 체계를 고장전의 상태로 복구할수 있다.

대역망접근은 HPS를 통하여 TCP/IP와 UDP/IP를 지원함으로써 실현된다. 단일파일체계는 망파일체계(NFS) 또는 안드레이파일체계(AFS)를 통하여 제공될수 있다. SP는 망파일체계들(NFS 또는 AFS)외에 대역적인 디스크접근을 위한 가상공유디스크(Virtual share disk, VSD)를 제공한다.

VSD는 NFS보다 더 좋은 성능을 가진다. VSD는 AIX론리볼륨관리기(Logical Volume Manage)의 최상위에 위치하는 장치구동프로그람층이다. 마디의 프로쎄스가 국부적으로 접속된 공유디스크에 접근하려고 하면 VSD는 그 요청을 마디의 LVM에 직접 보낸다. 프로쎄스가 원격공유디스크에 접근하려고 하면 VSD는 HPS를 통하여 자료를 직접 원격디스크의 VSD로 보낸다. 그다음 원격디스크의 VSD는 자료를 원격마디의 LVM에로 보낸다.

체계관리

SP체계console은 하나의 조종워크스테이션(그림 10-6)이다. SP체계관리자는 이 단일 조종점으로부터 전체 SP체계를 관리한다. 관리기능들은 다음과 같다.

- 체계의 설치와 감시,구성
- 체계동작
- 사용자관리
- 파일관리
- 일감회계
- 인쇄와 우편봉사들

그 외에 SP2하드웨어의 매 마디와 교환기, 프레임(frame)은 환경조건들을 알아 내고 하드웨어구성요소들을 조종하는 하나의 관리기판을 가진다. 관리기는 이 기판을 사용하여 전원의 투입/차단, 감시, 개별적인 마디의 재설정, 구성요소들의 교체를 진행할수 있다.

부하관리

SP는 2가지 형태의 사용자일감 즉 호상작용방식과 묶음방식을 지원한다. LSF는 또한 호상작용 및 묶음일감의 부하를 다 관리하는 SP에도 리용할수 있다.

10. 3. 5. SP2과 미래

1994년의 첫 출현으로부터 SP2체계구조는 많은 방향에서 개선되였다.
표 10-7은 SP2체계들의 1996년도 판본의 구조속성들에 대한 실례를 보여 준다.

표 10-7 1997년도 IBM SP2의 구조속성들

속성	604 높은 마디	넓은 마디
처리기수	2~8 Power PC 604	하나의P2SC
처리기당 최대속도	112MHz 224Mflop/s	125MHz, 540Mflop/s
처리기당 캐쉬	16KB 코드 16KB자료 L1캐쉬 또는1MB L2캐쉬	32KB코드, 128KB 자료 L1캐쉬, L2캐쉬는 없음
기본RAM기억기	64MB~2GB	64MB~2GB
기억기모선폭	256b	256b
내부디스크용량	2.2~6.6GB	2.2~36.4GB
Micro Channel조절자수	14	7
하나의 표준체계에서 최대마디수	16	1~8
한방향 점대점 최대통신대역너비	150MB/s	150MB/s

2개의 마디형태 즉 새로운 넓은 마디와 SMP마디가 그림 10-7의 본래의 좁은 마디, 좁은 마디 2, 넓은 마디에 보충된다.

새로운 마디형태

새로운 넓은 마디는 단일한 POWER2초고속소편(P2SC)처리기로 구성된다. P2SC처리기는 본래의 8개 소편 POWER2 처리기를 단일한 소편으로 통합하고 160KB의 내장캐쉬와 15000000개의 3극소자를 포함한다. 처리기박자주파수도 66.7MHz로부터 135MHz로 높아진다.

SMP마디(high node라고도 한다.)는 2～8개의 박자가 112MHz인 PowerPC 604처리기들로 구성된다. 매 604처리기는 하나의 32KB 소편내장 1차캐쉬와 하나의 1MB소편외장캐쉬를 가진다.

특별한 SP구성

상업용구성에서 SP체계는 16개까지의 PowerPC SMP마디들을 포함하여 128개까지의 마디들을 가진다. 전용구성에서 기계크기는 512개 마디로 늘일수 있다.

HPS의 기술은 다중교환기망구조를 사용하지만 련결대역너비는 40MB/S로부터 150M/S로 개선된다. 이것과 정합시키기 위하여 HPS에로의 마디 I/0대면부의 대역너비는 80MB/S로부터 160MB/S로 개선된다.

쏘프트웨어개선

쏘프트웨어의 개선은 보충적인 2개의 체계쏘프트웨어부분을 포함한다. 고유용성클라스터다중처리(High Availability Cluster Multi Processing, HACMP)는 쏘프트웨어의 복제와 고장회피유용성기능들을 제공한다. 병렬체계지원프로그람(Parallel Systems Support Programs)쏘프트웨어는 사용자와 암호관리, 일감회계, 체계감시, 체계분할과 같은 체계관리를 헐하게 한다.

IBM/LLNL Blue Pacific

에네르기집단은 IBM 및 Lawrence Livermore국가연구소와 테라flops초고속콤퓨터를 만들데 대한 계약을 체결하였다. 이 체계는 11.3에서 론의한다. 명백히 그것은 SP(Scalable Parallel)기술을 더욱 확장할것이다. 체계가 현재의 SP구조보다 우수하게 단일체계영상과 분산기억기연산들에 대한 더 많은 지원을 가짐으로써 클리스터개념을 더욱더 추구할것이라는것이 추측된다.

10. 4. Digital의 TruCluster

DEC TruCluster는 Unix클라스터들중의 하나이다. 마디는 Digital Unix를 실행하는 단일처리기 또는 다중처리기일수 있다. 호상접속망은 GigaSwitch 또는 기억기통로일수 있다. 이 부분에서는 AlphaSever 8400마디들과 기억기통로호상접속을 사용하는 TruCluster를 론의한다.

１０．４．１．TruCluster의 구성방식

TruCluster의 구성방식을 그림 10-13에서 보여 준다. 이 구조는 공유 없는 방법들과 공유디스크방법들을 결합한다.

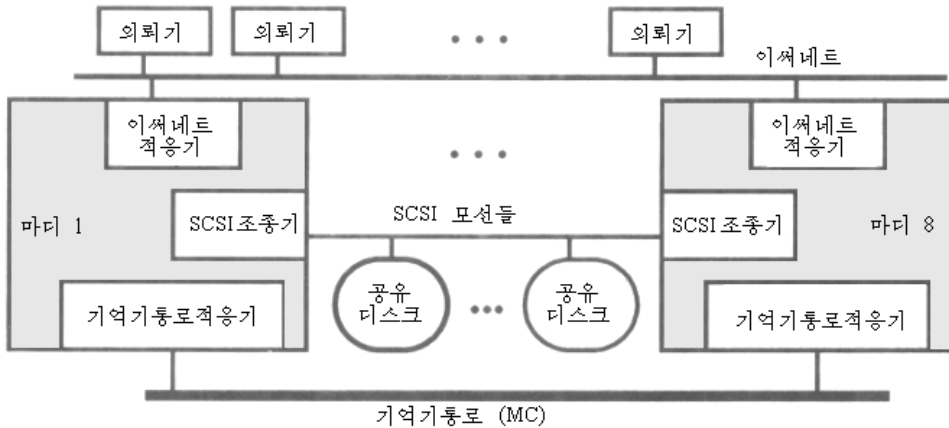



그림 10-13. TruCluster망의 구조

TruCluster는 콤퓨터 또는 자료기지봉사기로 설계된다. 이것은 SMP마디들과 혁신적인 기억기통로호상접속, 효률적인 통신과 선택된 단일체계영상, 쏘프트웨어의 유용성 및 확대가능성지원을 위한 반사기억기기구로 특징 지어 진다. 서로 다른 구성들은 서로 다른 사용자의 요구를 충족시킨다. 실례로 공유디스크들은 공유 없는 구조를 창조하도록 개별적인 마디들에로 분산될수 있다.

기억기통로

그림 10-13의 클라스터체계는 3개의 분리된 클라스터폭(Cluster wide)의 호상접속들을 가진다. 이써네트는 의뢰기워크스테이션들과 PC들, 말단들을 TruCluster에 접속하는데 사용된다. 사용자의 가입과 명령들, 응용들은 자기의 표준망으로부터 TruCluster에 접근한다. 마디가 공유디스크들의 파일들에 접근할수 있도록 하기 위하여 여러개의 SCSI모선들이 사용된다. 매 마디는 자체의 국부디스크들을 가질수 있다.

Encore Computer Systems와 Digital Equipment가 공동개발한 기억기통로(MC)는 높은 속도를 가지며 통보문넘기기와 동기화에 대한 지원을 담당한다. 이 구성은 이전에 처리하지 못했던 다음의 문제들을 평가하는데 쓸모 있다.

- MC와 관련된 쏘프트웨어를 어떻게 사용하면 동기화와 통보문넘기기를 위한 공유기억기(반사기억기)의 제한된 형식을 실현하는가?
- 분산자물쇠관리기를 어떻게 사용하면 디스크공유와 대등하게 하겠는가?
- SMP마디들의 클라스터를 어떻게 프로그람작성하는가?

TruCluster의 현재의 구성들은 8개 포구기억기통로로 접속된 8개까지의 마디들을 가질수 있다. TruCluster체계의 구조파라메터들을 표 10-8에서 보여 준다. 파라메터들은 모두 확대가능성의 한계를 보여 주는 최대값들이다. 그것들은 동시적으로 달성되지 않는다.

표 10-8　　DEC TruCluster의 구조파라메터들

구조파라메터	하나의 마디	하나의 TruCluster
처리기의 수	12	96
최대속도	10Gflop/s	80Gflop/s
소편외장캐쉬	8MB	64B
최대기억기	28GB	224GB
기억기모선 집합대역너비	2.1GB/s피크, 1.8GB/s유지	16.8GB/s최대 14.4GB/s유지
내부디스크	192GB	1.5TB
I/O통로들	12개PCI모선	96개 PCI모선들
I/O확장홈들	144개PCI확장홈들	1152
총 I/O대역너비	1.2GB/s	9.6GB/s
마디 대 마디최대대역너비	N/A	100MB/s

마디구조

매 마디는 DEC AlphaServer 8400 SMP체계이다.

구조적특징들은 표 8-1에서 보여　준다. AlphaServer의 블로크도식은 그림 10-14에서 보여　준다. AlphaServer 8400마디는 9개 접속홈기억기모선을 가진다. 매 접속홈은 하나의 CPU모듈 또는 기억기모듈, I/0모듈을 유지할수 있다. 하나의 마디는 적어도 하나의 CPU모듈과 하나의 기억기모듈, 하나의 I/0모듈로 구성되여야 한다. 6개까지의 CPU모듈과 7개까지의 기억기모듈, 3개까지의 I/0모듈이 있을수 있다.

CPU모듈은 2개의 437MHz Alpha 21164처리기를 포함한다. 매 처리기는 소편에 16KB의 1차캐쉬 1개와 96KB의 2차캐쉬 1개를 가진다. CPU모듈에 설치된 매 처리기에 하나의 4MB 외장형3차캐쉬가 있다. 기억기모듈은 4GB 4통로끼움식SIMM기억기와 함께 있을수 있다.

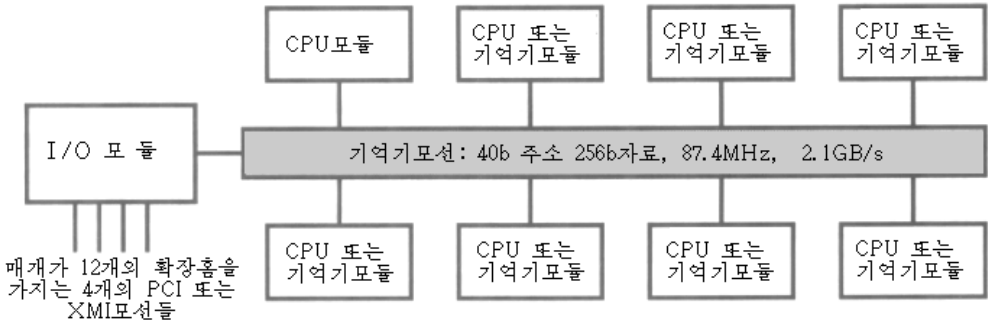



그림 10-14. Alpha Server 8400의 블로크도식

I/O모듈은 매개가 12개 확장홈을 가지는 표준 PCI모선 또는 Digital XMI모선의 4개의 I/O통로들을 조종한다. PCI모선과 XMI모선의 결합은 허용된다. 144개까지의 I/O확장홈들이 SCSI디스크와 CD-ROM구동장치, 이써네트적응기들과 같은 여러가지 표준I/O장치들을 접속한다.

기억기통로의 적응기는 PCI의 확장홈에 접속된다.

기억기모선은 CPU박자주파수의 1/5인 87.4MHz로 박자화된 동기모선이다. 모선자료폭은 256b 또는 32B이다. 최대대역너비는 32× 87.4=2796MB/S가 한계이다. 내재적인 하드웨어부가처리들을 고려하면 최대대역너비 즉 절대로 릉가할수 없는 대역너비는 2GB/s이다. 기억기모선은 대역너비를 1.8GB/s까지 확대하기 위하여 분산트랜잭션과 분리된 주소/명령모선 및 자료모선과 같은 기술들을 리용한다.

10. 4. 2. 기억기통로호상접속

기억기통로는 믿음성 있고 강력하며 효률적인 클라스터호상접속을 제공하기 위하여 설계된다. 특히 그것은 다음의 설계요구를 가진다.

- **높은 속도** 100GB/s의 1통로통보문대역너비와 1.5μs이하의 지연, 0.5μs이하의 처리기부가처리
- **오유처리** 작업중교체(hot-swap)능력과 망통신오유로부터의 오유검사 및 회복지원
- **PCI접속성** 공업표준 PCI모선에로의 대면부

5μs이하의 통보문지연 특히 500ns의 처리기부가처리를 달성하기 위하여 통신쏘프트웨어는 완전히 사용자수준에서 실행되여야 하고 통신규약은 령복사규약이여야 하며 쏘프트웨어는 검사합계산을 하지 말아야 한다.

왜냐하면 현재의 단일처리기가 SMP마디들에서 체계호출에만도 5μs이상 소비할수 있으며 특히 짧은 통보문들에 대하여 기억기복사와 검사합은 비용이 많이 들기때문이다. 이것은 사용자수준의 통신을 위하여 호출되며 기억기통로만을 개별적인 마디들의 가상주소공간으로 넘기고 하드웨어가 오유처리를 잘하도록 한다. DEC는 Encore의 반사기억기술을 기억기통로의 기초로 확장하였다.

반사기억기(reflective memory)

반사기억기의 개념은 그림 10-15에서 설명한다.

반사기억기는 다음과 같이 동작한다. 접속은 한 마디의 가상주소페지로부터 다른 마디의 가상페지로 기억기통로주소공간을 통하여 확립된다. 실례로 접속 2는 마디 3의 가상페지 E를 마디 1의 가상페지 B와 마디 2의 가상페지 C, 마디 2의 가상페지 D, 마디 3의 가상페지 F, 마디 4의 가상페지 I에 접속한다. 마디 2가 페지 E에 쓰면 기억기통로하드웨어는 쓴 내용들을 페지 E에 접속된 모든 가상페지에 자동적으로 넘기기한다. 다시 말하여 국부가상페지에로의 쓰기는 접속된 모든 국부 또는 원격가상마디에 반사된다. 후에 마디 1이 페지 B를 읽을 때 마디 2에 의하여 페지 E에 쓴 자료를 가져 온다. 반사기

억기는 다음과 같은 특징들을 가진다.

- 첫째로, 그것은 가상주소에 기초한다. 자료는 한 가상페지로부터 다른 가상페지로 직접 통신한다. 이것은 령복사를 실현하며 페지표를 유지하고 주소변환을 수행하는 기억기통로적응기들을 요구한다.
- 둘째로, 그것은 사용자수준통신기구이다. 접속시동은 핵심부호출을 요구한다. 그러나 비용이 많이 드는 접속확립은 한번만 수행되면 된다. 같은 접속은 그 어떤 보충적인 체계호출을 요구함이 없이 많은 통신을 수행하는데 사용될수 있다.
- 셋째로, 통신립도는 32B 자료블로크수준에 있다. 접속이 페지수준에서 확립되여도 자료통신들은 32B 블로크들로 수행된다. 하드웨어는 전체 페지를 한번에 전송하지 않아도 된다.
- 넷째로, 통신들은 보통의 store와 load지령에 의하여 수행된다. 사용자가 통보문넘기기쏘프트웨어로 통신할수 있다 해도 사용자는 그렇게 하려하지 않는다.
- 다섯째로, 접속들은 점대점(접속 1) 또는 방송(접속 2), 집단내방송(접속 3)일수 있다.
- 여섯째로, 접속들은 본질적으로 한방향이다. 하나의 가상페지는 전송과 수신을 동시에 할수 없다. 한 마디가 같은 접속으로부터 송신과 수신을 요구하면 그것은 분리된 2개의 가상페지를 사용하여야 한다.
- 일곱째로, 하나이상의 마디들이 같은 접속에 송신하는것(실례로 접속3)은 허용한다. 기억기통로쏘프트웨어는 자료일치성을 담보하기 위한 동기화명령들을 제공한다.

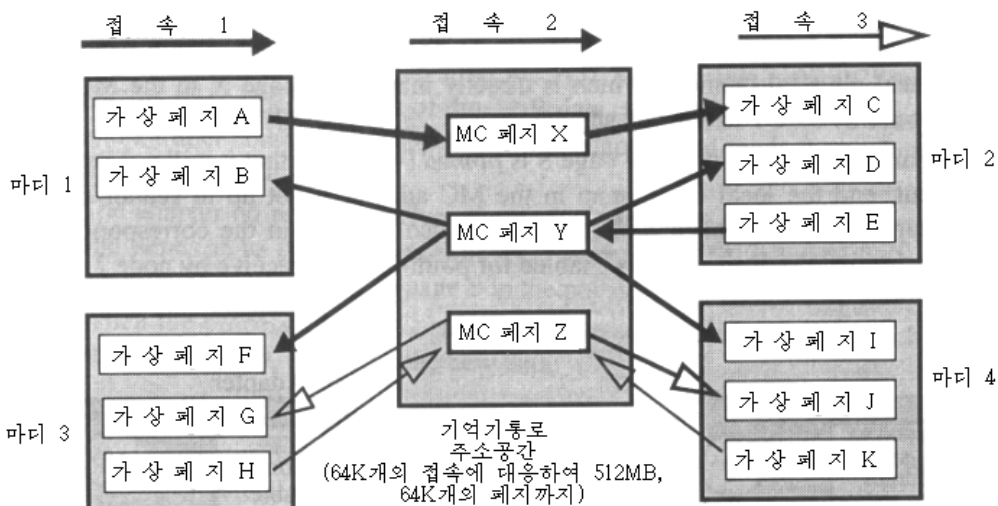



그림 10-15. 기억기통로사용에서 반사기억기개념의 설명

기억기통로를 통한 통신

접속이 어떻게 확립되며 32B 자료블로크가 어떻게 통신되는가를 더 상세히 보자. 기

억기통로는 마디의 프로쎄스들이 자기의 가상주소공간들의 페지를 넘길수 있는 512MB의 대역주소공간(그림 10.15의 MC주소공간)을 지원한다. 매 마디에서 이 MC주소공간은 기억기통로적응기의 512MB PCI I/O주소공간창문으로 직접 넘겨 진다. TruCluster는 8KB의 페지크기를 가진다. 이것은 기억기통로가 64K개의 서로 다른 접속들을 확립하는데 64K개의 페지들을 제공할수 있다는것을 의미한다.

그림 10.15의 접속 1이 어떻게 확립되는가를 아래에서 설명한다. 그림 10.16을 참조하면 접속 1을 확립하는데 다음과 같은 걸음들이 실행된다.

- 응용은 클라스터봉사서고의 함수를 호출하며 이 함수는 접속을 위하여 MC주소공간의 한개 페지(페지 X)를 배정한다.
- 마디 1의 프로쎄스(송신마디)는 자기의 주소공간의 가상페지 A를 페지 X에 련결하기 위한 서고함수를 호출한다. 이 함수는 가상페지 A를 I/O공간의 물리페지 Q로 넘기는 마디의 가상 대 물리넘기기표(map)에 하나의 입력자료를 만든다. 이것은 MC적응기의 페지조종표(page control table, PCT)에 의하여 MC주소공간의 페지 X로 직접 넘겨 진다. 이 함수는 또한 접속이 가능한 대응하는 PCT입력자료에서 마디 1에 의한 점대점통신을 가리킨다.
- 마디 2(수신마디)의 프로쎄스는 자기의 주소공간의 가상페지(페지 E)를 페지 X에 련결하기 위한 서고함수를 호출한다. 이 함수는 가상페지 A를 국부물리기억기의 페지 S로 넘기는 마디의 가상 대 물리넘기기표에 하나의 입력자료를 만든다. 이것은 MC적응기의 PCT에 의하여 MC주소공간의 페지 X로 직접 넘겨 진다.

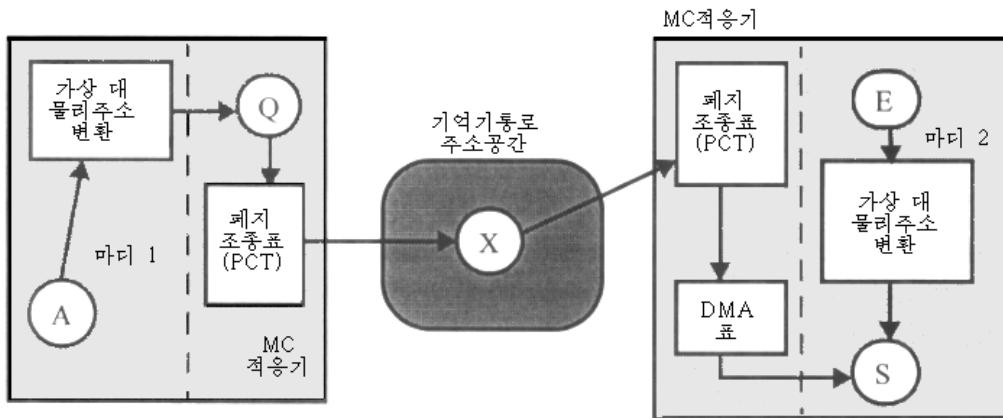



그림 10-16. 기억기통로의 통신기구

- 더우기 이 물리페지 S는 바뀌지 않도록 고정되며 MC적응기의 물리DMA표는 접속 X로부터 페지 S에로 자료를 산란(scatter)시키도록 만들어 진다. 이 함수는 또한 접속이 가능한 대응하는 PCT입력자료에서 마디 2에 의한 점대점수신을 가리킨다.

페지조종표(PCT)

접속의 PCT입력자료는 다음과 같은 속성들과 동작들을 기록하는데 사용되는 여러개

의 조종비트들을 포함한다.

- 전송 또는 수신가능
- 방송 또는 집단내 방송, 점대점
- 국부복사만들기
- 모든 수신마디들에서 쓰기도착에 대한 하드웨어응답
- 쓰기가 목적지마디에 도착한후 목적지마디를 차단

블로크자료전송

접속 X가 확립된후 마디 1로부터 마디 2에로 32B 자료블로크를 전송하기 위하여 다음의 걸음들이 실행된다.

- 마디 1의 프로쎄스는 가상페지 A의 상대주소에 련속적인 쓰기(기억)들을 수행하며 이것은 물리페지 Q로 변환된다. Q는 I/O공간이므로 쓰기들은 모든 캐쉬들을 무시한다. 그러나 쓰기들은 32B 쓰기완충기들중의 하나에서 통합된다. 처리기가 쓰기완충기를 flush하면 가득찬 32B 쓰기는 PCI모선으로 전송되며 MC적응기가 선택된다.
- 적응기는 PCI의 쓰기를 하나의 MC파케트로 교갑화하기 위하여 PCT의 정보를 사용한다. MC파케트는 32B의 자료와 목적지를 식별하는 하나의 머리부, 32b CRC를 포함하는 하나의 꼬리부로 구성된다. 그다음 적응기는 그 파케트를 망으로 내보낸다.
- 마디 2의 적응기는 파케트를 받고 MC머리부와 꼬리부를 벗긴 다음 PCI모선에 보낸다. PCT와 DMA표는 들어 오는 자료가 물리기억기의 페지 S로 가는것이라는 것을 결정한다.
- 다음에 마디 2의 프로쎄스가 가상페지 E(물리페지 S를 가리킨다)를 읽으면 그것은 새로운 자료를 얻는다.

기억기통로하드웨어는 많은 오유처리 및 흐름조종능력들을 제공한다. 하드웨어는 매우 낮은 전송오유률을 가질뿐아니라 모든 통보문쓰기에 대한 자동오유발견도 할수 있다. 그것은 활성교환지원과 하드웨어에 기초한 충분한 흐름조종을 제공한다. 그것은 오유조건에서도 엄격한 통보문쓰기순서를 진행한다. 하드웨어는 또한 하나의 고속자물쇠명령을 제공한다.

10.4.3. TruCluster프로그람작성

TruCluster프로그람작성을 위한 쏘프트웨어구조를 그림 10-17에서 설명한다. 말단사용자병렬응용들은 표준C나 Fortran, 고성능 또는 MPI서고함수들을 호출한다. 이 서고들모두는 만능통보문넘기기(Universal Mess Fortran(HPF)언어들을 사용하여 개발되였으며 병행과 통보문넘기기를 특정화하기 위하여 PVM age Passing, UMP)층의 최상위에서 실현된다.

UMP의 기본기능은 기억기통로와 SMP마디의 공유기억기구조, 미래의 통신 및 호상접속들에서 사용될수 있는 통일적인 통보문넘기기 API를 제공하는것이다.

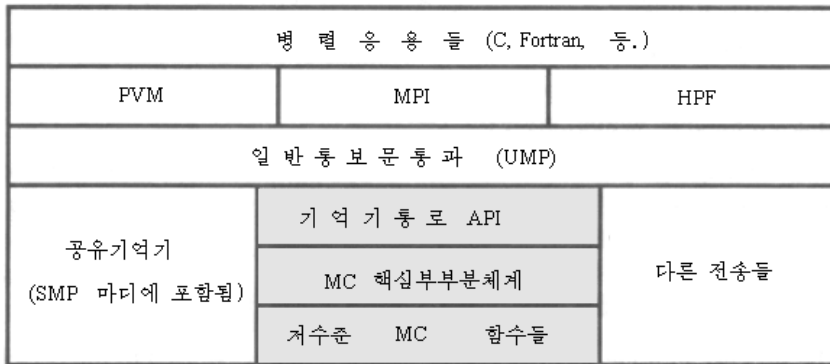



그림 10−17. TruCluster 프로그람작성을 위한 쏘프트웨어구조

기억기통로쏘프트웨어지원

검은 통은 기억기통로쏘프트웨어이다. 그것은 하나의 사용자수준MC API와 핵심부수준쏘프트웨어의 2개 층을 포함한다. 말단사용자들은 PVM이나 MPI, HPF만을 사용하여야하며 낮은 층들은 사용하지 말아야 한다.

체계프로그람작성자들은 물론 UMP를 사용해도 된다. MC API는 지어 3부류체계쏘프트웨어개발자들에게도 너무 저수준이다.

MC API는 12개의 함수들로 구성된다. 4개의 함수들은 MC접속들을 창조하고 파괴하는데 사용된다. 또 다른 4개 함수들은 대역MC공간에서 회전(spin)자물쇠들을 관리하는데 사용된다. 2개는 기억기통로오유들을 검사하는데 사용된다. kill함수는 클라스터원격마디의 특정한 프로쎄스에 Unix신호를 보낸다. 마지막으로 클라스터의 마디수와 호스트이름들을 얻는 함수가 있다.

통신이 보통 store/load지령들에 의하여 실현되므로 송신/수신함수들은 없다는데 주의하여야 한다.

UMP층은 표 10−9에 기록한바와 같이 12개의 함수들을 포함한다. 프로그람작성자의 관점에서 UMP는 TCP와 비슷하다. UMP의 연산들은 통로(TCP에서는 접속이라고 한다.)라고 부르는 2중 점대점 련결들에 기초하고 있다. 통로는 1개 프로쎄스쌍사이의 쌍방향통신을 제공하기 위하여 사용된 한방향완충기들의 쌍이다. 이 프로쎄스들을 **통로의 끝점**이라고 부른다. 2개의 프로쎄스는 클라스터의 같은 마디에 상주할수도 있고 클라스터의 서로 다른 마디들에 상주할수도 있다.

통로들과 UMP연산들은 단일마디의 공유기억기공간이나 MC주소공간에서 실현될수 있다. 공유기억기실현은 더 좋은 성능을 가지면서도 UMP의 사용자들에게 투명하다.

프로쎄스는 한개 과제쌍사이의 통로를 열기 위하여 ump_open을 호출한다. 그러나 두번째 프로쎄스가 또 같은 통로를 열 때까지 그 통로는 확립되지 않는다. 첫번째 프로쎄스는 두번째 프로쎄스에 의하여 통로접속이 완성되였다는 사건을 기다리기 위하여 ump_listen과 ump_wait를 사용할수 있다.

표 10-9 일반통보문넘기기함수들

함수	의미
ump_init	UMP를 초기화하고 자원들을 배정한다.
ump_exit	UMP를 닫고 자원들을 해방한다.
ump_open	2개의 프로쎄스끌점들사이의 2중UMP통로를 연다.
ump_close	UMP통로를 닫는다.
ump_isten	한 통로를 위한 한개 끌점을 등록한다.
ump_wait	UMP사건이 일어 나기를 기다린다.
ump_read	통보문을 통로로부터 읽는다.
ump_wirte	통보문을 통로에 쓴다.
ump_nbread	통로로부터 특정한 량의 통보문을 읽는다.
ump_nbwrite	Ump_write의 비차단판본
ump_incast	집단내 방송:통보문을 통로들의 목록에 쓴다.
ump_info	UMP의 구성과 상태정보를 되돌린다.

이것은 TCP의 listen함수와 같다. ump_wait함수는 융통성이 있으며 통보문의 도착, 통로의 지우기 등과 같은 다른 사건들을 기다릴수 있다. 함수 ump _ read와 ump_nbread, ump_write, ump_nbwrite, ump_mcast들은 확립된 통로에서 통보문을 보내고 받는데 사용된다.

TruCluster의 여러 통보문층들의 성능을 표 10-10에서 보여 준다. 이식가능한 MPI실현 MPICH와 DEC실현의 PVM, MPI를 포함한다.   여기서 SM은 마디내에 공유기억기를 가지고 있다.

표 10-10 DEC TruCluster의 여러 통보문층들에서 점대점통신의 성능

통보문층	지연 t_0(us)		대역너비 r_∞(MB/s)		반최대길이 $m_1/2$(Byte)	
	SM	MC	SM	MC	SM	MC
MC API	N/A	2.9	N/A	64	N/A	186
UMP	2	5.8	75	61	150	354
DEC MPI	5.2	6.4	64	61	333	421
DEC PVM	3	8	66	63	148	344
MPICH	30	N/A	24	N/A	720	421

다른 클라스터들과 MPP기계들에 비하여 TruCluster는 기억기통로호상접속에서 8μs 또는 그 이하의 작은 통신지연을 가진다. 대역너비값들은 MPI와 PVM을 가장 낮은 수준의 쏘프트웨어(MC API)가 제공할수 있는것에 가깝게 실현될수 있다는것을 보여 준다.

같은 마디의 프로쎄스들은 공유기억기구조의 우월성으로 하여 더 효률적으로 통보문을 보낼수 있다.

１０．４．４．TruCluster체계쏘프트웨어

TruCluster체계는 공유SCSI모선들을 통하여 접속한 국부마디디스크와 대역디스크를 다 포함한다(그림 10−13을 참고). TruCluster체계쏘프트웨어(Cardoza et al..[127]을 참고)는 마디들을 감시하고 구성요소고장사건의 회복절차들을 자동적으로 초기화함으로써 고유용성을 지원한다. 그것은 병렬자료기지를 지원하여 자료기지응용들이 장치가 국부장치이건 원격장치이건 상관없이 생디스크와 다른 I/O장치들에 동시에 접근할수 있게 한다. 그것은 PVM과 MPI, HPF확장들을 가지는 C와 Fortran어로 작성된 병렬프로그람들을 지원한다.

TruCluster체계쏘프트웨어는 다음과 같은 요소부분체계들을 포함한다.

분산자물쇠관리자(Distributed Lock Manager, DLM) 클라스터폭의 접근이 디스크파일들과 같은 공유자원들과 동기를 맞추기 위한 쏘프트웨어서고함수들을 제공한다.

접속관리자 클라스터성원의 행로 즉 얼마나 많은 마디들이 그 클라스터를 구성하는가를 보존한다. 그것은 또한 클라스터마디들의 모든 쌍사이의 통신경로를 확립하고 유지한다.

분산생디스크(Distributed Raw Disk, DRD) 생디스크에 기초한 사용자수준의 응용(실례로 분산자료기지관리체계, 트랜잭션처리모니터)은 클라스터의 어디에 물리기억기가 있는가에 상관없이 클라스터에서 실행할수 있게 한다. 이것은 응용들이 다수의 마디들로부터 RAID디스크들을 포함하는 기억매체에 병렬로 접근할수 있게 한다.

대역오유기록기(Global Error Logger) TruCluster쏘프트웨어가 하나이상의 체계들에로의 TruCluster환경에서 일어 나는 사건들에 대한 통보문들을 기록하게 하여 TruCluster관리자들이 그러한 사건들이 일어 날 때 중요한 문제들에 대한 통지를 받을 수 있게 한다.

클라스터모니터 체계관리자의 그라픽스사용자대면부이다. 클라스터모니터는 클라스터에서 유용성과 접속성의 현 상태와 같은 클라스터구성을 현시하는데 사용될수 있다. 관리자는 단일위치에서 전체 클라스터체계를 관리하기 위하여 관리도구들을 호출할수 있다.

개선된 POLYCENTER파일체계(Advanced File System, AdvFS) 이것은 유용성을 지원하기 위한 실행기록식국부파일체계이다. 트랜잭션실행기록을 사용하여 AdvFS는 예상밖의 전원상태로 하여 재시동후에 시간단위가 아니라 초단위로 파일령역들을 회복할수 있다.

１０．５．Berkeley NOW 프로젝트

Berkeley 캘리포니아대학의 NOW(워크스테이션망)과제는 전용클라스터에도 응용할수 있는 기업클라스터기술개발을 목적으로 하고 있다.

이 기술들은 큰 규모의 콤퓨터를 구성하기 위하여 작고 대량생산된 상업체계들을 리용한다. 동기와 목적은 클라스터식체계가 개별적인 사용자에게 전용워크스테이션에 대한 빠르고 예측가능한 응답시간을 제공하며 너무 커서 탁상콤퓨터가 클라스터전면에 걸쳐 자원들을 증강해야 하는 과제들도 허용하는것이다.

NOW 과제는 9장에서 론의한 대부분의 클라스터문제들을 처리한다. 효률적인 통신은 상업적기가비트망들과 능동적인 통보문통신규약을 리용함으로써 지원된다. 단일체계영상과 자원관리, 유용성은 CLUnix라고 부르는 사용자수준의 클라스터폭쏘프트웨어를 통하여 제공된다. XFS라고 하는 봉사기 없는 망파일체계는 확장가능하고 고도로 유용한 단일파일계층을 지원하기 위하여 개발되였다. 최근에 NOW집단은 고도로 유용하고 증가적으로 확장가능하며 지리적으로 떨어 져 있는 Web봉사들을 구성하기 위한 WebOS라고 불리우는 쏘프트웨어골조(framework)를 구성하고 있다.

１０．５．１．고속통신을 위한 능동통보문

능동통보문은 낮은 부가처리를 가지는 통신을 실현하기 위한 비동기통신기구이다. 그 목적은 밑에 놓여 있는 기본통신하드웨어의 본래능력을 사용자에게 보여 주는것이다.

기본착상은 통보문머리부의 조종정보를 통보문조종기라고 하는 사용자수준부분루틴에로의 지시자로 사용하는것이다. 통보문머리부가 목적지마디에 도착하면 통보문조종기가 호출되여 망으로부터 나머지통보문을 추출하여 그것을 진행중인 계산과 통합한다.

능동통보문기구

능동통보문기구는 하나의 일반적인 명세서[179]와 서로 다른 실현들을 가진다. 보충적인 기능들을 제공하기 위하여 최초의 능동통보문을 확장할데 대한 제기들도 있다. 여기서는 일반능동통보문판본 1의 기본기구만을 론의한다. 왜냐하면 그것이 대부분의 실현에 따르는 명세서이기때문이다. 판본 2[426]는 최근에 나왔는데 널리 실현되지 않고 있다. 판본 1의 명세서는 그림 10-18에서 보여 주는바와 같이 적은 수의 함수(명령)들에 대한 사용을 규정한다. 이 함수들은 다음과 같은 가정에 기초한다.

- 능동통보문은 SPMD응용의 프로쎄스들사이의 통신들을 지원하는데 사용되는 하나의 쏘프트웨어층이다.
- 프로그람은 매개가 0부터 n−1까지의 가상마디번호(VNN)로 배렬된 *n*개의 처리기들로 구성된다. 프로쎄스의 창조와 관리는 능동통보문층밖의 어떤 체계봉사(실례로 GLUnix)에 의하여 제공된다.

일반능동통보문명세서에 따라 콤파일하는 능동통보문층을 GAM층이라고 부른다. 이

층은 그림 10−18에서 보여 주는 5개의 기본명령(연산)들을 제공한다. 여기서 vnn_t와 handler_t는 각각 가상마디번호와 통보문조종기부분루틴에 대하여 GAM가 정의한 자료형이다. GAM은 여러개의 유틸리티함수들도 제공한다. GAM에는 2가지 형태의 통보문 즉 아래에서 정의하는 요청통보문과 응답통보문이 있다.

요청통보문(Request Message)

요청통보문 W는 다음의 요청함수를 호출하여 보낸다.

am _ request_ 2 (Destination, request_handler, x, y)

```
기본함수들 :
int am_request_M(vnn_t dest, request_handler, int arg0, ..., int argM-1)
int am_reply_M(vnn_t dest, reply_ handler, int arg0, ..., int argM-1)
int am_get(vnn_t source, void *lva, void *rva, int nbytes,
                handler_t reply_handler, void *handler_arg)
int am_store(vnn_t dest, void *rva, void *lva, int nbytes,
                handler_t request_handler, void *handler_arg)
void am_poll(void)

유틸리티함수들 :
int am_enable(...)          // 능동통보문층을 초기화한다.
int am_disable(void)        // 능동통보문층을 탈퇴한다.
int am_procs(void)          // 프로그람의 총 처리수를 되돌린다.
int am_my_proc(void)        // 호출하는 처리의 가상마디번호를 되돌린다.
int am_max_size(void)       // get/store에 대한 최대바이트수를 되돌린다.
```

그림 10−18. 일반능동통보문의 일부 함수들

여기서 요청통보문 W는 request_handler와 2개의 정수 x와 y, 요청하는 프로세스의 가상마디번호로 구성된다. 요청명령 am_request_M(⋯)은 M=0부터 M=4까지의 5개 판본을 가지며 매 판본은 M개의 정수인수들을 가진다. 우의 요청은 2개의 인수 x와 y를 가진다. 이 요청은 W를 Destination프로세스로 보낸다. am_request는 W가 보내질 때까지 귀환되지 않도록 차단된다. W가 도착하면 Destination의 프로세스는 인수 x와 y, 요청프로세스의 VNN을 가지는 부분루틴 request_handler를 호출한다.

응답통보문(Reply Message)

응답연산은 요청통보문과 매우 비슷하다. 응답통보문은 다음의 함수를 호출하여 보

낸다.

am_reply_2(Destination, reply_handler, x, y)

여기서 응답통보문 W는 reply_handler와 2개의 정수 *x*와 *y*, 응답하는 프로쎄스의 가상마디번호로 구성된다. 응답명령 am_reply_M(⋯)은 M=0부터 M=4까지의 5개 판본을 가지며 매 판본은 M개의 정수인수들을 가진다. 우의 응답은 2개의 인수 *x*와 *y*를 가진다. 이 응답은 W를 Destination의 프로쎄스로 보낸다. am_reply는 W가 보내질 때까지 귀환되지 않도록 차단한다. W가 도착하면 Destination의 프로쎄스는 인수 *x*와 *y*, 응답프로쎄스의 VNN을 가지는 reply_handler부분루틴을 호출한다.

통보문조종기들은 빠르며 미리 결정된 시간경계내에 있도록 되여 있다. 다시 말하여 조종기함수들은 절대로 지연되지 않는다. 이 목표를 달성하기 위하여 GAM은 다음과 같은 제한들을 명기한다.

- 응답함수는 오직 하나의 요청조종기부분루틴으로부터 호출되여야 한다.
- 응답함수의 Destination은 오직 요청하는 처리일수 있다.
- 응답조종기부분루틴은 요청이나 응답함수를 호출할수 없다.

am_stor와 am_get함수들은 2개의 프로쎄스사이의 대규모자료전송에 사용된다. am_store함수는 다음과 같이 정의된다.

am_store(Dest, lva, rva, N, request_handler, handler_arg)

이 함수는 호출자(요청하는 프로쎄스)의 시작주소 lva로부터 N바이트의 련속적인 기억령역을 Dest프로쎄스의 rva로 시작하는 기억령역에 전송한다. N바이트의 자료를 모두 받은 다음 Dest프로쎄스는 request_handler를 호출하고 그것을 파라메터인 요청프로쎄스의 VNN와 rva, N, handler_arg에로 통과시킨다. am_get는 am_store와 비슷한데 방향이 반대이다.

am_get(Source, rva, lva, N, reply_handler, handler_arg)

함수 am_get호출은 호출자(응답하는 프로쎄스)의 시작주소 rva로부터 N바이트의 련속적인 기억령역을 Source프로쎄스의 lva로부터 시작되는 기억기령역에로 꺼낸다. N바이트의 자료를 모두 받은 다음 호출자(source는 아니다.)는 reply_handler를 호출하고 그것을 파라메터인 source프로쎄스의 VNN과 lva, N, handler_arg에로 통과시킨다.

실례 10.4. NOW에서 능동통보문의 발생

그림 10-19에서 보여 주는바와 같이 짧은 통보문통신에 GAM을 사용하는 2개 프로쎄스프로그람을 고찰하자. 프로쎄스 Q는 배렬 A를 계산하고 프로쎄스 P는 스칼라값 x를

계산한다. 최종결과는 x와 A[7]의 합이다. P가 Q로부터 A[7]를 원격으로 꺼내서 x와 국부적으로 합하면 된다.

이 원격꺼내기를 실현하기 위하여 프로세스 P는 하나의 am_request를 실행하여 프로세스 Q에 첨수값 7을 보낸다. 요청통보문을 받은 다음 Q는 요청조종기함수 request를 호출한다. request_h함수의 귀환은 reply함수를 호출하여 A[7]의 값을 P로 돌려 보낸다.

프로세스 *P*

```
compute x
...
am_enable(...) ;
am_request_1(Q, request_h, 7);
am_poll();
sum = x+y ;
...
am_disable();


int reply_h(vnn_t Q, int z)
{ y = z; }
```

프로세스 *Q*

```
compute A
...
am_enable(...) ;
...
am_poll();
...
am_disable();


int request_h(vnn_t P, int k)
{ am_reply_1(P, reply_h, A[k]); }
```

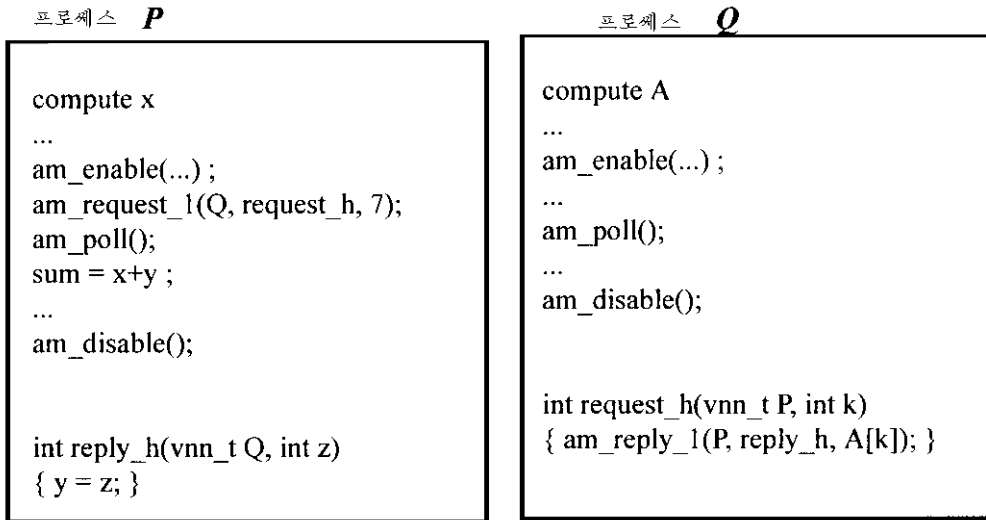



그림 10−19. NOW의 GAM에 의한 원격꺼내기의 실현

응답통보문을 받은 다음 P는 응답조종기함수 reply_h를 호출하며 reply_h는 A[7]를 P의 변수 Y로 통과시킨다. 함수 am_poll은 임의의 도착통보문들을 위하여 통신망을 등록한다.

다음의 실례는 2개의 워크스테이션마디들상의 큰 블로크자료전송과 같은 긴 통보문의 통신에 어떻게 GAM을 사용하는가를 보여 준다.

실례 10.5. GAM층을 사용하는 긴 통보의 통신

그림 10−20에서 보여 주는바와 같이 프로세스 P는 N바이트의 배렬 A를 계산하고 그 값을 프로세스 Q의 배렬 B에 기억한다. 불필요한 완충 및 복사부가처리를 피하기 위하여 먼저 request_reply를 통하여 프로세스 Q의 기억기령역 B가 자료를 받기 위하여 준비되여 있다는것을 확인하기 위한 대화가 수행된다. 하나이상의 요청(또는 응답)조종기가 있을수 있다. 어느 조종기가 사용되는가는 요청(또는 응답)함수호출에 명기된다.

- P가 보내는 첫번째 요청통보문(am_request_0)이 request1 조종기를 호출하며 request2 조종기는 응답통보문을 통하여 기억기령역 B의 시작주소를 P로 되돌린다.
- P가 보내는 두번째 요청통보문(am_store)은 request2를 호출하며 request2는 대량자료가 전송된후에 실행된다.

이 실례에서 request2는 빈(null)함수이다.

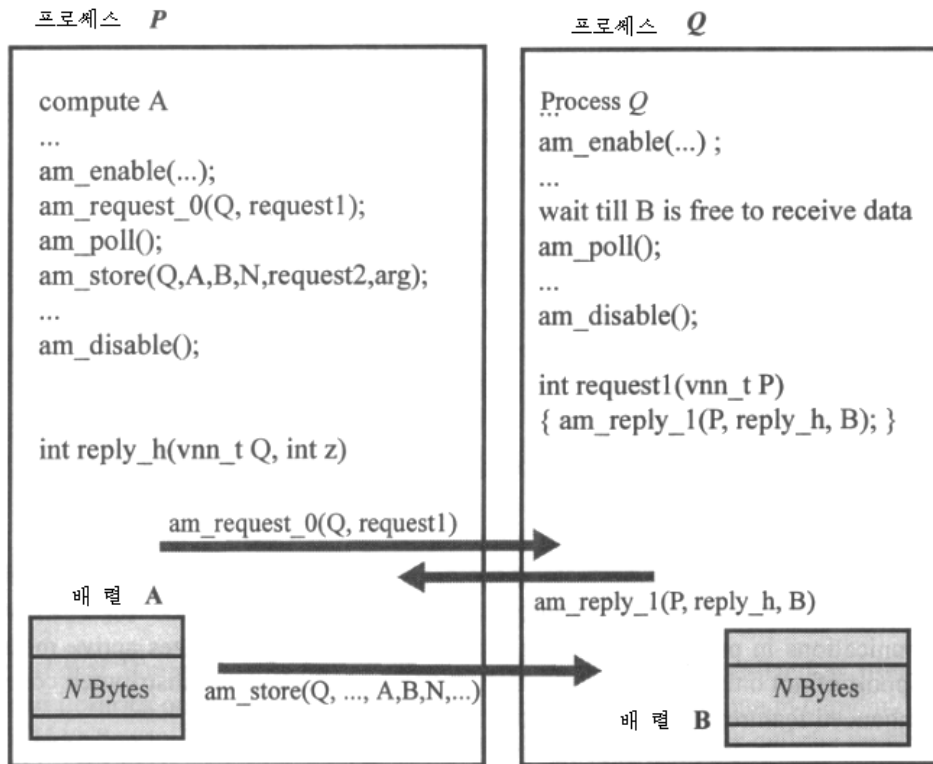



그림 10-20. GAM에 의한 큰 자료블로크의 대량전송

능동통보문의 실현

능동통보문은 일반적인 기구이므로 임의의 하드웨어 또는 쏘프트웨어가동환경에 종속될것을 요구하지 않는다. 그것은 MPP들과 워크스테이션들의 클라스터, 지어는 PVM에서도 실현된다. 그러나 능동통보문은 통신하드웨어의 본래성능의 큰 몫을 달성할수 있는 저수준통신층으로 실현된다.

이 목표는 지연과 대역너비의 국면에서 만족되였다. 실례로 FDDI로 접속한 HP9000/735워크스테이션들의 클라스터견본에서 능동통보문은 대량자료전송을 위한 충분한 련결대역너비와 175B통보문에 대하여 최대대역너비의 절반을 달성한다. 시동시간은 25us로서 작으며 크기에 있어서 TCP의 성능보다 더 좋다.

능동통보문의 좋은 성능은 다음의 원인들에 의한것이다.

- **사용자수준** 능동통보문통신들은 문맥절환 및 보호경계교차와 관련된 부가처리를 없애기 위하여 흔히 전체가 사용자공간에서 실현된다. 임의의 체계호출을 내보낼 필요가 없다. 사용자의 응용은 망대면부하드웨어에 직접 접근할수 있다. 통보문조종기는 보통 사용자수준의 부분루틴이다.
- **단순성** 일반능동통보문들은 매개가 매우 단순한 기능과 규약을 가지고 있는 5개의 명령들만을 가진다. 실례로 그것들은 TCP와 같은 규약들에서 부가처리의 주요원천인 임의의 완충엔진리나 오유검사를 하지 않는다.

- **계산과 통신의 겹침** 통보문쏘프트웨어(실례로 PVM, MPI)는 송신측과 수신측모두에서 통보문완충을 요구하는 비차단송신/수신연산들을 통하여 겹침을 제공한다. 능동통보문은 짧은 통보문과 긴 통보문을 구별하여 취급한다. 4개 또는 그이하의 단어들로 된 짧은 자료블로크는 차단 am_request에 의해 전송될수 있다. 대량자료전송들은 am_store 또는 am_get를 사용한다. 일반능동통보문은 통신과 계산의 겹침을 쉽게 하기 위하여 am_store의 비차단판본을 제공한다.

GAM−2

일반능동통보문명세서의 판본 2는 보충된 많은 기능과 함께 최근에 발표되였다[426]. GAM은 병렬계산프로그람들에서 낮은 부가처리의 통신을 제공하기 위한것이다. GAM−2는 능동통보문을 병렬콤퓨터지원뿐아니라 망 및 분산콤퓨터지원에로 일반화한다. 개선된 특징들은 다음과 같다.

- GAM은 병렬계산프로그람들만 지원한다. GAM3는 보충적으로 MPMD와 망, 분산, 의뢰기, 봉사기의 응용들을 지원한다.
- GAM은 단일스레드의 프로쎄스들만을 지원한다. GAM2는 다중스레드의 프로쎄스들도 허용한다.
- GAM은 고장안전의미론만을 지원한다. 즉 한 프로쎄스가 중지하면 전체 병렬프로그람이 중지되거나 죽는다. GAM−2는 고장허용 및 고유용성을 가능하게 한다.

１０．５．２．대역자원관리를 위한 GLUnix

클라스터들은 유용성과 단일체계영상을 제공하기 위하여 일반적인 워크스테이션OS에는 없는 새로운 조작체계기능들을 요구한다.

두가지 문제가 해결되여야 한다.

정확히 어떤 기능이 보충되여야 하는가?

필요한 기능들이 어떻게 보충되여야 하는가?

조작체계확장

Vahdat et al..[622]은 많은 조작체계에서 혁신이 있었지만 상업조작체계에는 거의나 통합되지 않았다는것을 언급하였다. 그들은 조작체계는 그것의 기능에 의해서가 아니라 오히려 지원하는 응용들과 하드웨어 /OS 결합의 가격/성능, 체계의 건전성에 의하여 구입된다는것을 지적하였다. 그것들은 현재의 방법들을 평가하고 모두가 한계를 가진다고 결론하였다. 2개의 방법을 아래에서 론의하였다.

극소형핵심부　2개의 잘 알려 진 실례들은 Mach와 Windows NT이다. 모든 봉사들을 제공하는 단일핵심부는 여러 모듈들로 교체되고 대부분의 본질적인 봉사들은 극소형핵심부라고 하는 하나의 작은 모듈에 의하여 제공된다. 일부 봉사들은 사용자방식에서 제공된다. 모듈성은 융통성과 이식가능성을 제공한다. 실례로 Windows NT는 Windows 3

과 OS/2, Unix와 같은 서로 다른 조작체계들과 사용자수준경쟁을 할수 있다. 그러나 극소형핵심부방법은 여러 부족점을 가진다.

- OS모듈들(핵심부포함)의 초기판본은 림시기억기로부터 가능한껏 다시 쓸것을 요구한다.
- 서로 다른 하드웨어구성들을 일치시키기 위하여 요구되는 노력은 여전히 본질적이다.
- 극소형핵심부체계는 단일체계보다 더 높은 부가처리를 가진다. 왜냐하면 그것이 문맥절환과 보호경계교차, 프로쎄스호상간의 통신에 보다 활동성을 가지기때문이다. 사용자수준의 경쟁은 부가처리를 더욱 높인다.

사용자수준데몬과 서고 Condor와 PVM, LSF에서와 같이 새로운 기능들은 상업조작체계의 최상위에서 사용수준의 봉사와 서고들로서 실현될수 있다. 여기서 최상위는 사용자수준의 봉사들과 서고들이 밑에 있는 조작체계를 호출한다는것을 의미한다. 이 방법은 핵심부의 수정을 요구하지 않으며 실현하기 쉽다는 우월성을 가진다. 또한 극소형핵심부방법에서 본 문맥절환과 프로쎄스호상간의 통신에 대한 부가처리를 발생시킨다.

GLUnix방법

GLUnix는 대역층Unix를 위한것이다. 다음과 같은 2개의 층으로 구성되여야 한다.

- 낮은 층은 핵심부방식(핵심부수준)에서 실행하는 마디들의 상업조작체계이다.
- 높은 층은 클라스터에 요구되는 새로운 특징들을 제공하는 사용자수준조작체계이다.

특히 이 대역층은 클라스터의 마디들에 대한 단일체계영상을 제공하여 처리기와 기억기, 망용량, 디스크대역너비모두가 순차 및 병렬응용들에 배정될수 있도록 한다. 대역층은 동적으로 만든 응용에 련결되는 보호방식의 사용자수준조작체계서고로서 실현된다.

서고는 모든 체계호출을 차단하며 그것을 가능한껏 응용주소공간의 절차호출로서 실현한다. 사용자수준의 보호는 쏘프트웨어고장분리(software fault isolation , SFI)방식[630]에 의하여 가능하다. 이것은 목적코드에서 모든 기억 및 분기명령앞에 검사를 삽입함으로써 언어독립형으로 실현된다. 이 방법은 많은 클라스터특징들이 사용자수준에서 실현될수 있게 한다. GLUnix방법은 여러가지 우월성을 가진다.

- **실현하기 쉬움** GLUnix는 사용자수준에서 완전히 실현되며 핵심부의 수정을 요구하지 않는다. 첫번째 GLUnix견본은 3달만에 실현되였다.
- **이식성** 이식성은 대부분의 상업조작체계들에 있는 기반체계로부터 추출한 표준적인 특징들의 최소모임에 기초하고 있다. GLUnix는 프로쎄스사이의 통신과 프로쎄스의 신호만들기, 부하정보에로의 접근을 지원하는 임의의 조작체계에 이식가능하다.
- **효률성** 클라스터에 요구되는 새로운 특징들은 응용주소공간내에서 절차호출에 의해 호출된다. 하드웨어보호경계와 교차를 요구하지 않으며 핵심부트래프(trap)

나 또는 문맥절환을 요구하지 않는다. 여러 마디들에서 여러 GLUnix복사들을 동등하게 하는것은 공유기억기토막들이나 프로쎄스사이의 통신명령들을 사용하여 실현될수 있다.

- **건전성** GLUnix가 사용자수준에 있으므로 그것은 전통적인 결함수정도구들을 사용하여 철저히 검사될수 있다. 오유는 발견되고 진단되며 제거될수 있다.

GLUnix의 특징

GLUnix의 견본은 기업클라스터들에서 중요하다고 생각되는 다음과 같은 대부분의 특징들을 제공하도록 실현되였다.

- 병렬프로그람들의 협조적인 일정짜기
- 휴식자원의 발견과 프로쎄스의 이동, 부하평형
- 고속사용자수준통신
- 원격페지화
- 유용성지원

GLUnix견본은 LSF에서 찾아 볼수 있는것과 비슷한 유틸리티도구들의 모임을 제공한다. GLUnix의 착상은 대단히 흥미를 끄는것 같다. 사용자수준의 착상이 클라스터들에 요구되는 모든 기능들을 제공하는데서 충분한 융통성을 가지는가 하는것과 SFI의 부가처리가 더 빠른 문맥절환에 의해 보상되는가하는 시간만이 증명할것이다.

10. 5. 3. 봉사기 없는 xFS파일체계

봉사기 없는 파일체계는 파일봉사기의 기능들을 클라스터의 모든 마디들에 분산시킨다. 전통적인 집중파일봉사기모형은 그림 10-21에서 봉사기 없는 모형과 대비된다.

ㄱ) 중앙파일봉사기를 가지는 망파일체계

ㄴ) NOW의 봉사기없는 망파일체계, xFS

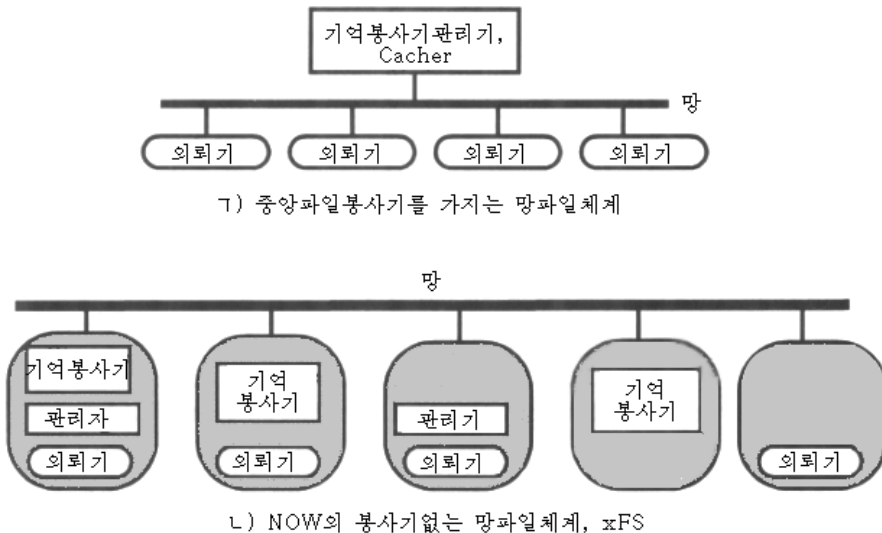



그림 10-21. 두가지 형태의 파일체계모형의 비교

- **중앙기억** 자료파일들과 메타자료는 파일봉사기에 련결된 하나 또는 그이상의 안정디스크들에 기억된다. 파일의 메타자료는 파일형태(정규파일, 등록부, FIFO, 련결, 장치 등), 파일크기, 장치 ID(여기에 파일이 배치된다.), 마디번호, 파일소유자 ID, 파일접근허가 등과 같은 파일속성들의 모임으로 이루어 진다.

- **중앙캐쉬** 성능을 개선하기 위하여 매 의뢰기는 일부 파일블로크들을 국부적으로 캐쉬화한다. 그외에 파일봉사기는 의뢰기의 일부 실패들을 충족시키기 위하여 자기의 주기억에 자주 사용되는 파일블로크들을 캐쉬한다.

- **중앙관리** 봉사기는 메타자료와 캐쉬일관성에 대한 관리를 포함하는 모든 파일체계관리기능들을 수행한다. XFS(그림 10-2 ㄴ)에서 모든 봉사기 및 의뢰기의 기능들은 분산방식으로 모든 마디들에 의하여 수행된다. 실례로 하나의 마디는 기억봉사기, 의뢰엔진리자모두가 동작할수 있다(가장 왼쪽마디에서 본것).

XFS에서 유일한 제한은 관리자가 의뢰기대면부를 사용하므로 관리자처럼 동작하는 마디도 의뢰기여야 한다는것이다. XFS에는 아직 기억봉사기들이 있지만 그것들은 클라스터의 마디들에 분산되며 단일한 파일봉사기에 집중되지 않는다.

값 눅은 디스크여유배렬(Redundant Array of Inexpensive Disks, RAID)

그림 10-21 ㄴ) 의 봉사기 없는 파일체계는 값 비싼 하드웨어RAID를 사용함이 없이 고성능과 고유용성을 제공한다. 최근에 xFS는 단일기우성디스크스트리핑(stripping)을 사용한다. 한 파일의 자료블로크들은 여러 기억봉사기마디들로 스트리프(strip)되며 다른 마디에 기우성블로크를 첨부한다. 마디가 고장나면 고장난 디스크의 내용들을 남은 디스크들과 기우성디스크의 배타합을 실시하여 재구성할수 있다.

RAID의 부족점은 작은 쓰기문제이다. 즉 만일 하나의 쓰기가 전체 스트리프가 아니라 한개의 스트리프의 일부분만 수정하면 새로운 기우성디스크를 추정하기 위하여 낡은 기우성디스크와 일부 낡은 자료를 읽어야 한다. 많은 작은 쓰기가 수행된다면 이것은 큰 부가처리를 발생시킨다.

XFS는 이 문제를 풀기 위하여 기록식스트리핑을 사용한다. 매 의뢰기는 먼저 쓰기들을 각자의 의뢰기기록에 합친다. 의뢰기실행기록은 모든 쓰기들을 실행기록하는 기억기완충기이다. 그 다음기록은 기록토막들을 사용하는 디스크들로 보내진다. 매 토막은 $k-1$개의 실행기록쪼각들로 이루어 지며 기우성쪼각과 함께 모두 k개의 기억봉사기들에 보내진다.

실례 10.6. 쏘프트웨어 RAID를 사용하는 XFS의 기록식스트리핑

클라스터는 4개의 기억기봉사기들을 가지며 xFS를 받아 들인다. 그림 10-22에서 보여 주는바와 같이 의뢰기가 디스크들을 2개의 기록토막으로 위임하는(commit) 기록에로 쓰기들을 합친다고 하자.

그림 10−22. 쏘프트웨어 RIAD의 xFS를 사용하는 기록식스트리핑

첫번째 토막은 3개의 쪼각 1, 2, 3으로 구성되며 두번째 토막은 3개의 쪼각 4, 5, 6으로 구성된다. 기우성쪼각 $p(q)$는 쪼각 1, 2, 3(4, 5, 6)의 배타합으로써 추정된다. 이 2개의 토막들은 4개의 기억봉사기들에 스티리프형식으로 기억된다.

우의 기록식스트리핑은 큰 다중봉사기클라스터들에 대하여 다음의 2가지 문제점을 가진다.

첫째로, 의뢰기는 많은 작은 쪼각들을 기억봉사기들로 보내야 한다. 큰 토막들을 쓰기 위하여서는 기록기억기가 대단히 커야 한다.

둘째로, 여러 의뢰기들이 자기의 기록들을 쓰려고 하면 그것들은 봉사기를 놓고 경쟁해야 한다.

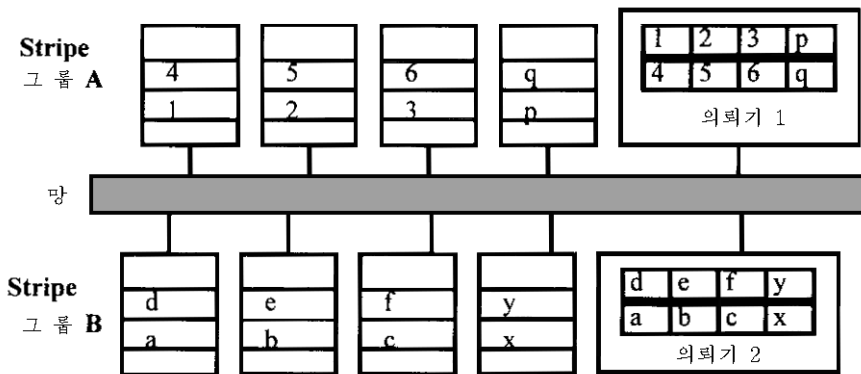



그림 10−23. 분산봉사기들에서 xFS를 사용하는 스트리프무리들

XFS에서 이 문제들은 기억봉사기들을 스트리프무리(group)구룹이라고 하는 부분모임들로 나누는 방법으로 해결된다. 그림 10−23에서는 8개의 봉사기들이 매개가 4개의 봉사기들을 가지는 2개의 무리로 나누인다. 쪼각크기는 단일한 8개 봉사기무리를 사용하는것보다 2.33배 크다. 의뢰기 1과 의뢰기 2는 자기의 기록들을 충돌이 없이 2개의 무리들에 동시에 쓸수 있다.

스트리프무리들은 아래와 같은 우월성들도 가진다. 부족점은 지금의 클라스터당 하나의 기우성디스크대신에 매 무리에 하나의 기우성디스크가 있다는것이다.

협조적인 파일캐쉬

협조적인 파일캐쉬의 착상은 단순하다[182]. 즉 클라스터의 매 의뢰기마디는 기억기의 한 부분을 파일캐쉬로 배정한다. 협조적인 캐쉬알고리듬은 큰 클라스터폭의 파일캐쉬를 만들기 위하여 이 모든 기억기들을 리용한다.

의뢰기가 국부파일캐쉬실패를 만나면 그것이 전통적인 워크스테이션에서 진행되기때문에 디스크로 가지 않고 자료를 꺼내기 위하여 다른 의뢰기의 기억기로 간다. 원격기억기의 사용은 표 10-11에서 설명된다.

표 10-11 원격기억기 또는 디스크로부터 8KB의 블로크에 접근하는 시간

망매체	10Mbps 이써네트		155Mbps ATM	
기억	원격기억기	원격디스크	원격기억기	원격디스크
기억기복사	250 us	250 us	250 us	250 us
시동시간	400 us	400 us	400 us	400 us
자료전송	6250 us	6250 us	400 us	400 us
디스크	0	14800 us	0	14800 us
전체 시간	6900 us	21700 us	1050 us	15850us

이 표는 원격기억기 또는 원격디스크로부터의 읽기에 의한 국부캐쉬실패에 드는 평가시간들을 보여 준다. 국부디스로부터의 읽기는 같은 디스크시간(14800μs)이 걸린다. 자료는 원격기억기읽기가 이써네트에서보다 3배 더 빠르지만 ATM보다는 14배 더 빠르다는것을 보여 준다. 디스크에 비한 원격기억기의 속도우세는 기억기복사와 망의 속도가 디스크속도보다 더 큰 비률로 빨라 지기때문에 계속 커질것이다.

그림 10-24에서 보여 주는바와 같이 파일은 봉사기의 디스크에 안정하게 기억된다. 전통적으로 파일들은 의뢰기의 기억기(대부분의 파일체계들)와 의뢰기의 국부디스크(실례로 AFS), 봉사기의 기억기(대부분의 파일체계들)에서 캐쉬되였다. 또한 xFS는 파일이 동료의뢰기들의 (원격)기억기들에서 캐쉬되도록 한다. 여러개의 협조적인 캐쉬알고리듬들이 제안되였다[182]. 아래에서는 2개의 협조적인 캐쉬알고리듬들 즉 greedy forwarding 또는 N-chance forwarding을 론의한다.

Greedy Forwarding

이 방법은 다음과 같이 동작한다.

파일에 접근하는 의뢰기가 먼저 국부캐쉬를 시도한다. 자료블로크가 거기에 없으면 의뢰기는 요청을 봉사기로 보낸다. 봉사기는 다음과 같은 2가지 결과를 가지고 자기의 국부캐쉬를 탐색한다.

- 성공하면 봉사기는 자료를 의뢰기로 돌려 보낸다. 봉사기는 또한 매 블로크수준에서 모든 의뢰기의 캐쉬내용들을 기록하는 캐쉬등록부를 유지한다.

- 실패하면 봉사기는 자기의 캐쉬등록부를 참고하여 요청된 자료를 캐쉬가 가지고

있는 의뢰기로 요청을 보낸다. 그다음 이뒤의 의뢰기는 자료를 직접 요청한 의뢰기로 보낸다. 자료가 의뢰기들의 캐쉬에 다 없으면 봉사기는 자기의 디스크로부터 자료를 얻어 목적지의뢰기로 넘기기한다.

이 방법의 부족점은 같은 블로크가 많은 캐쉬들에 중복된다는것이다. 의뢰기가 첫번째로 블로크를 읽을 때 블로크는 봉사기의 디스크로부터 꺼내 져 두개의 장소 즉 봉사기캐쉬와 의뢰기캐쉬에서 캐쉬된다. 같은 블로크를 읽는 다른 의뢰기는 그것을 자기의 국부파일캐쉬에 캐쉬한다.

이 중복은 협조적인 캐쉬의 유효크기를 줄여 실패률을 증가시킨다. 더우기 중복은 캐쉬일관성관리를 더 힘들게 한다.

그러나 이 방법은 실현하기가 간단하다. 봉사기의 캐쉬등록부는 블로크당 약 24byte의 기억기를 가진다. 8KB의 캐쉬블로크들에 대하여 이것은 단지 0.3%이다. 6MB의 등록부는 2GB의 협조적인 캐쉬에 충분하다.

그림 10−24. 의뢰기-봉사기클라스터에서 파일을 캐쉬하는 서로 다른 방법들

N−Chance Forwarding

두번째 파일캐쉬알고리듬은 N기회보내기라고 하는 탐욕보내기의 일반화이다. 이것은 블로크를 오직 하나의 의뢰기캐쉬에서 캐쉬함으로써 중복문제를 피한다. 그러한 블로크를 singlet라고 한다. 의뢰기가 다른 의뢰기의 캐쉬로부터 하나의 블로크를 꺼내면 그 블로크는 두번째 캐쉬에서 비여 지며 통보문이 봉사기로 보내 져 블로크가 이동했다는것을 알린다.

Singlet에서의 문제는 보다 최근의 자료블로크를 위하여 만일 singlet가 캐쉬의 자기자리를 떠나 버려 지면 singlet는 전체 협조적인 캐쉬로부터 없어 진다.

N기회보내기는 이 문제를 다음과 같이 완화시킨다. 방법은 의뢰기캐쉬가 다 차지 않는 한 탐욕보내기로서 정확히 동작한다. 임의의 의뢰기가 다 찬 국부캐쉬를 만나 블로크버리기를 요구하면 의뢰기는 먼저 그 블로크가 singlet인가 검사한다. 만일 아니면 버리기는 간단히 그 블로크를 버린다. 블로크가 signlet이면 의뢰기는 블로크의 순환계수기를 N으로 설정하고 블로크를 임의의 의뢰기로 보내 거기서 캐쉬시킨다. 두번째 의뢰기가 후에 그 블로크를 버려야 하면 순환계수기를 감소시켜 그 블로크를 다른 의뢰기로 보낸다. 이 프로쎄스는 순환계수기가 0으로 될 때까지 계속된다. 그 다음블로크는 단순히 버려진다. 만일 그 블로크가 의뢰기에 의해 참조되면 그 블로크의 순환계수기는 N으로 재설

정되며 순환프로쎄스가 반복된다.

대응하는 캐쉬등록부입력자료는 새로운 의뢰기를 지시하도록 갱신되여야 한다. 순환계수기 N이 0(탐욕보내기와 등등)으로부터 1로 증가하면 성능은 25% 개선된다는것이 알려 져 있다.

N=1로부터 N=2로 되면 성능은 단지 5% 개선된다. N을 더 증가시키면 개선이 없다.

분산관리

xFS는 메타자료와 캐쉬일관성관리를 포함하여 파일체계관리기능들을 충분히 분산시킨다. 가장 중요하게 xFS는 여러 마디들에 위치하는 여러 관리자들을 받아 들인다. 어떻게 분산관리가 달성되는가를 보기 위하여 그림 10−25에서 보여 주는것과 같은 xFS의 단순화한 파일읽기동작을 살펴 보자.

의뢰기가 파일블로크를 읽으려 하면 파일의 경로이름과 파일안의 상대주소를 가지는 요청을 내보낸다. XFS의 매 등록부는 그자체가 하나의 파일이다. 파일의 이름을 사용하여 의뢰기는 파일의 어미등록부로부터 파일의 색인번호를 찾을수 있다.

그림 10−25. xFS의 단순화한 파일접근동작

색인번호는 유일하게 하나의 파일을 식별한다. 의뢰기는 색인번호를 자주 접근되는 파일블로크들을 기억하는 국부캐쉬(Unix 캐쉬)에서 탐색에 사용한다. 자료블로크가 그 캐쉬안에 있으면(캐쉬성공) 그 블로크는 꺼내진다. 그리하여 그러한 리상적인 경우들에는 모든 동작들이 국부적이며 디스크접근이 없이 수행된다. 캐쉬가 실패하면 의뢰기는 어느 물리마디가 색인번호들중 어느 무리를 관리하는가 즉 정확한 관리자는 어디에 있는가를 가리키는 자료구조인 관리자표를 조사하는데 그 색인번호를 사용한다. 이 표는 파일이 창조될 때 갱신된다. 또한 관리표가 모든 관리자와 의뢰기에 사본되므로 관리표에서 찾을수 있는 국부연산이다.

그다음 정확한 관리자가 같은 마디에 없으면 색인 및 상대주소정보는 망을 통하여

정확한 관리자에게 보내진다. 응답가능한 매 파일에 대하여 관리자는 블로크가 캐쉬되는가 또는 디스크에서 그 블로크의 정확한 위치, 캐쉬한 블로크가 일치하는가와 같은 정보의 행로를 보존한다.

블로크가 같은 의뢰기의 국부캐쉬에서 캐쉬되고 일치되면 관리자는 읽기요청을 의뢰기로 보내며 의뢰기는 자기의 국부캐쉬에서 자료블로크를 꺼내여 그것을 직접 본래의 의뢰기로 보낸다.

자료블로크가 협조적인 캐쉬에 없으면 관리자는 색인마디(또는 inode)를 찾기 위한 imap라는 자료구조를 조사한다. imap는 파일의 모든 자료블로크들의 주소들을 포함한다. 요구하는 자료블로크의 주소는 자료블로크가 꺼내지고 요청의뢰기로 보내지는 정확한 기억봉사기를 찾는데 사용한다.

캐쉬일관성은 매 블로크별로 무효와 후쓰기규약으로 실현된다. 블로크를 수정하려는 의뢰기는 먼저 그 블로크에 대한 쓰기소유권을 얻기 위하여 파일의 관리자에게 통보문을 보내야 한다. 관리자는 임의의 다른 캐쉬복사들을 무효시키고 캐쉬상태를 새로운 소유자로 갱신하여 소유권을 그 의뢰기에게 준다. 일단 의뢰기가 쓰기소유권을 가지면 매번 소유권을 얻지 않고 여러번 블로크에 되풀이하여 쓸수 있다. 의뢰기는 다른 의뢰기가 같은 자료블로크를 읽거나 쓸 때까지 쓰기소유권을 유지한다. 다음 관리자는 소유권을 해제하고 임의의 변화들을 안정기억에 flush한 다음 자료블로크를 새로운 소유자에게 보낸다.

NFS 대 xFS의 성능

쓰기성능을 개선하기 위하여 xFS는 기록식파일체계(logged file system, LFS)를 사용한다. LFS는 지우기를 요구하며 프로쎄스가 파일들에로의 가입들을 통합하기 위하여 쪼개진 디스크공간을 다시 정렬할것을 요구한다.

xFS는 지우기가 병목으로 되지 않도록 분산병렬지우기방법을 실현한다. xFS를 위한 집합대역너비는 파일체계를 동시에 읽고 쓰는 의뢰기들이 점점 보충됨에 따라 계속 높아진다. xFS대역너비는 표 10-12에서 보여 주는것과 같이 읽기에 대해서는 13.8MB/s, 쓰기에 대해서는 13.9MB/s에 이른다.

그러나 NFS는 적은 개선요인들만을 가지는 경우 5개의 의뢰기가 최대이다. xFS의 단일의뢰기성능은 NFS만큼 좋지 못하다.

Anderson et al..[34]는 이것이 xFS기술의 제한보다는 견본실현(실례로 핵심부봉사대신에 사용자수준의 대몬들을 사용하는것)때문이라고 믿고 있다.

표 10-12 32개 마디클라스터에서 NFS와 XFS의 비교

파일연산	NFS			XFS		
	1개 의뢰기	5개 의뢰기	개선	1개 의뢰기	32개 의뢰기	개선
대규모읽기	1.2MB/s	2.7 MB/s	2.25	0.9 MB/s	13.8 MB/s	15.33
대규모쓰기	1.2 MB/s	1.4 MB/s	1.17	0.6 MB/s	13.9 MB/s	23.17
소규모쓰기	22files/s	86 files/s	3.91	40 files/s	1122 files/s	28.05

작은 파일들에 대한 쓰기성능을 측정하기 위하여 매 의뢰기가 2048개의 1KB파일들을 창조하는 실행시간들을 측정한다.

xFS는 단일의뢰기의 경우에조차 더 좋은 성능을 보여 준다. xFS의 확대가능성은 NFS보다 더 좋다. NFS는 매초당 86개의 파일을 창조하는 총 처리량을 가지는 5개의 의뢰기가 한계이다.

10. 6. 쏘프트웨어실현된 DSM 클라스터 TreadMarks

Rice대학의 Zwaenepoel집단에 의하여 개발된 단일체계영상, 단일기억기공간, DSM실시간서고의 마지막측면으로서 TreadMarks를 조사한다. 이 체계는 워크스테이션들의 클라스터에서 분산공유기억기를 제공한다.

먼저 클라스터에서의 단일기억기공간제공에서 고려되여야 할 주요문제들을 상세히 본다. 다음 TreadMarks가 어떻게 이 문제들을 처리하는가를 론의한다. 그다음 TreadMarks 착상의 쏘프트웨어실현을 론의한다.

클라스터단일기억기공간을 얼마나 잘 지원하는가를 평가하는데서 다음의 문제들이 고려되여야 한다.

사용자대면부는 무엇인가?

어떻게 기억기일치성을 보존할수 있는가?

어떤 성능적특징들과 기술들이 제공되는가?

이 물음들에는 아래와 같이 대답이 주어 진다.

10. 6. 1. 경계조건

경계조건들은 DSM쏘프트웨어에 의한 제한들이다. 다음의것들은 그 제한들에 대한 리해를 도울수 있는 부분적인 검사목록이다.

- 조작체계가 핵심부의 수정을 요구하는가? 덜 엄격한 물음은 DSM의 실현이 OS원천코드를 알것을 요구하는가?
- DSM이 클라스터통신하드웨어로부터의 특별한 지원을 요구하는가?
- DSM이 기억엔진하드웨어로부터의 특별한 지원을 요구하는가?

우의 물음들중 임의의것에 대한 대답이 《예》이면 실현이 힘들며 이식성이 실질적인 문제로 된다. 실례로 조작체계가 수정되여야 한다면 많은 사람들이 생산체계를 만드는 위치에 있을수 없다. 상업용조작체계들에서 원천코드사용허가의 값이 비싸진다.

Linux와 같은 자유Unix는 견본응용들이 부족하며 공동의 세계에서 아직 신용되지 않는다. 자유Unix를 사용될수 있다해도 그 체계를 Windows NT클라스터로 이식할것을 요구한다면 어떻게 되겠는가?

다행히 TreadMarks는 이 3개의 질문들 모두에 《아니》하고 대답한다. TreadMarks는 Unix워크스테이션들의 클라스터의 사용자수준에서 실행한다. 그것은 핵심부수정이나 특

정한 특별취급을 요구하지 않는다. 그것은 표준Unix대면부와 콤파일러, Linker들을 사용한다. 결과 TreadMarks는 아주 이식가능하며 IBM과 DEC, SGI, HP, Sun Microsystems로부터 여러 Unix가동환경들에 이식되였다.

TreadMarks는 실시간서고로서 실현된다.

DSM이 요구하는 모든 통신들(실례로 페지복제, 기억기일치성연산들)은 sockets대면부를 통하여 표준UDP/IP를 사용하는 통보문넘기기에 의해 실현된다. 그것은 특정한 가동환경에 의해 제공되는 고성능하드웨어(실례로 DEC TurCluster의 기억기통로)의 우월성을 가질수 있지만 특정한 하드웨어지원을 요구하지 않는다. 그것은 Unix워크스테이션들의 이써네트클라스터에서 실행할수 있다.

１０．６．２．DSM을 위한 사용자대면부

단일기억기공간을 가지는 하나의 클라스터는 사용자들에게 하나의 SMP의 중앙공유기억기만을 마디의 여러 기억기들로 교체한 SMP와 같이 보인다. 다시 말하여 클라스터의 단일기억기공간은 다음의것들을 지원해야 한다.

- DSM서고에 대한 동적련결을 가지는 현재의 2진코드.

특히 순차 2진프로그람은 모든 클라스터마디들의 기억기들을 사용할수 있어야 한다. 이것은 아직까지 현재의 체계들에서 유효하게 해결되지 못한 문제이다.

- DSM서고에로 재콤파일되고 재련결(relink)되는 수정하지 않은 원천코드.
- 원천코드는 특정한 공유기억기구문들(서고의 루틴들, 콤파일러지령들 등)을 삽입하여 수정하여야 한다. 대부분의 쏘프트웨어 DSM들(Treadmarks 포함)은 이 방법을 따른다.

세번째 방법은 원천코드를 다시 쓸것을 요구하므로 DSM쏘프트웨어는 간단한 API를 제공해야 한다. TreadMarks API는 7개의 서고함수들만을 포함하므로 대단히 간단하다.

그림 10-26은 야코비(Jacobi)오락을 위한 TreadMarks프로그람을 보여 준다. 여기서 TreadMarks구문들은 첫번째 출현때 굵은 글자로 보여 준다. TreadMarks프로그람이 실행을 시작할 때 프로쎄스 0만이 실행되고 있다. Tmk_startup서고함수는 TreadMarks를 초기화하고 원격프로쎄스들을 시작한다. Tmk_nprocs변수는 병렬프로그람의 프로쎄스의 수이다. Tmk_proc_id변수는 0부터 Tmk_nprocs-1까지 정렬한 프로쎄스 ID를 포함한다. 공유기억기공간은 Tmk_malloc함수에 의해 배정된다. 공유공간은 함수 Tmk_free에 의해 해제된다.

정적으로 또는 malloc호출에 의해 배정된 기억기는 비공개이다. Tread Marks는 3개의 동기화함수를 제공한다. Tmk_barrier(k)는 장벽동기화를 위한것인데 여기서 k는 표식으로서 여기서는 무시될수 있다. 2개의 함수 Tmk _ lock _ acquire와 Tmk_lock_release는 각각 자물쇠를 획득하고 해제한다.

```
#define   M       1024
#define   N       1024
float     **grid;          /*  공유배렬  */
float     scratch[M][N]; /*  비공개배렬  */

main ( )
{
  Tmk_startup( );
  if ( Tmk_proc_id == 0 ) {
      grid = Tmk_malloc( M*N*sizeof(float));
      initialize grid ;
  }
  Tmk_barrier(0);
  length = M / Tmk_nrpocs;
  begin = length * Tmk_proc_id;
  end = length * (Tmk_proc_id+1);

  for ( number of iterations ) {
      for ( i = begin; i < end; i++ )
          for ( j = 0; j < N; j++ )
              scratch[i][j] = (grid[i-1][j] + grid[i+1][j] +
                              grid[i][j-1] + grid[i][j+1])/4;
      Tmk_barrier(1);
      for ( i = begin; i < end; i++ )
          for ( j = 0; j < N; j++ )
              grid[i][j] = scratch[i][j] ;
      Tmk_barrier(1) ;
  }
}
```

그림 10-26. TreadMarks C 코드에서 Jacobi 오락

10. 6. 3. 실현문제

TreadMarks는 공유기억기를 바이트들의 하나의 선형배렬로서 취급한다. TreadMarks는 워크스테이션마디들에서 현존 가상기억기하드웨어의 우월성을 가지는 페지수준자료공유를 사용한다. 마디의 국부기억기는 대역적으로 공유된 기억기의 캐쉬와 같다.

마디가 공유변수에 대한 읽기 또는 쓰기실패(페지고장)를 만나면 이 변수를 포함하는 페지는 TreadMarks에 의해 국부기억기에 복제된다. 읽기용페지들에 대한 복제는 아무런 문제도 일으키지 않는다. 그러나 페지가 마디에 의해 수정되면 TreadMarks실시간체계는 다른 마디들의 복제된 페지들을 무효시켜야 한다.

2가지 문제가 DSM의 효률을 크게 낮출수 있다.

첫번째 문제는 일치성을 실시하는것에 의해 생기는 통신부가처리이다.

두번째 문제는 2개의 상관 없는 변수(x와 y라고 한다.)가 같은 폐지에 배치되고 매 변수가 서로 다른 프로쎄스에 의해 써질 때의 거짓(false)공유이다. 2개의 프로쎄스는 x 또는 y를 공유하지 않지만 그 폐지를 공유한다.

TreadMarks는 이 문제들을 완화시키고 DSM을 효률화하기 위하여 3가지 기술을 사용한다. 그것들은 복제와 무효를 가지는 해제일치성과 일치성사건들에 대한 lazy프로쎄스, 다중쓰기규약이다.

해제일치성

해제일치성은 약한 일치성의 한 형식이다. 해제일치성의 착상은 다음의 관찰에 기초하고 있다. 병렬프로그람들은 정규적으로 끝나야 한다. 즉 병렬프로그람들은 자료경쟁을 가지지 말아야 한다. 만일 한 변수가 그 변수에 쓰기를 하는 여러 프로쎄스들에 의해 공유되여야 한다면 충돌접근이 불가능하도록 동기화연산들이 사용되여야 한다.

비공개 또는 읽기용변수에 대한 접근은 폐지의 다른 복사들을 무효화시키지 말아야 한다. 동기화는 같은 공유변수에 대한 2개의 충돌접근들사이에 제출되여야 한다(만일 접근들이 같은 공유변수에 접근하고 접근들중 적어도 하나가 쓰기이면 2개의 접근에 대한 재호출은 충돌하고 있다.). 그리하여 동기화연산이 끝날 때까지 프로쎄스는 공유변수의 수정에 대한 통지를 요구하지 않는다.

이것이 해제일치성의 본질이다. 즉 기억기일치성연산들은 공유변수가 수정된후가 아니라 자물쇠가 해제된후에 수행된다.

Lazy처리

해제일치성은 순차일치성모형에 필요한 통신을 줄이지만 아직도 불필요한 통신들을 수행한다. 실례로 그림 10-27의 코드에서 마지막 3개 행을 고찰하자. 프로쎄스 0이 Tmk_lock_release의 실행을 완료하는 첫번째 프로쎄스라고 하자. SUM은 모든 프로쎄스들에 복제되므로 통보문을 모든 프로쎄스들에 보내여 SUM이 변화됨을 알려야 한다.

실례 10.7. 공유기억기프로그람에서 3가지 변수형태

그림 10-27의 코드는 배렬 A의 합을 계산하는 TreadMarks프로그람을 설명한다. 3가지 변수형태 즉 매 프로쎄스에 국부적인 비공개변수(실례로 LocalSum)와 읽기용공유변수(실례로 배렬 A), 읽기/쓰기공유변수(실례로 Sum)가 있다.

마지막변수형태는 자료경쟁을 피하도록 자물쇠에 의해 보호되여야 한다. 그러나 이 코드토막은 림계구역(critical region)이므로 하나의 프로쎄스(말하자면 프로쎄스 1)만이 프로쎄스 0 다음에 그것을 실행할수 있다. 그리하여 하나의 프로쎄스만이 프로쎄스 0에 대한 SUM의 수정에 대하여 통지해 줄것을 요구한다. 통보문을 모든 프로쎄스에 보내는 것은 필요없으며 통신흐름은 증가한다.

TreadMarks는 이 문제를 푸는데 lazy해제일치성규약을 사용한다. 통지는 해제직후가 아니라 프로쎄스 1이 자물쇠를 획득할 때 보내진다.

```
#define   M        1024
float     *A, *Sum;        /* 공유변수 */
float     LocalSum = 0.0 ; /* 비공개변수 */
int       lock_id = 1;     /* 자물쇠변수 */

main ( )
{
   Tmk_startup( );
   if ( Tmk_proc_id == 0 ) {
        A = Tmk_malloc( (M+1)*sizeof(float));
        Sum = Tmk_malloc(sizeof(float));
        initialize array A; *Sum = 0.0; /* 공유변수
                                               초기화 */
   }
   Tmk_barrier(0);
   length = M / Tmk_nrpocs;
   begin = length * Tmk_proc_id;
   end = length * (Tmk_proc_id+1);

   for ( i = begin; i < end; i++ )
        LocalSum += A[i];

   Tmk_lock_acquire(lock_id); /* 자물쇠얻기 */
        *Sum += LocalSum; /* critical section */
   Tmk_lock_release(lock_id); /* 자물쇠해제 */
}
```

그림 10-27. 자물쇠를 가지는 TreadMarks 프로그람

TreadMarks는 통지를 프로세스 1에 보내기 위해서만 이 정보를 사용한다. 이 통지정보는 자물쇠허가통보문과 함께 넘기기될수 있으므로 통지를 위한 통보흐름은 완전히 제거된다. 그러므로 프로세스 1이 자물쇠를 획득하면 TreadMarks는 공유변수림계구역에 대한 이전의 실행에서 수정되고 SUM을 포함하는 그 페지에 대한 프로세스 1의 복사가 무효되였다는것을 프로세스 1에 알린다. 나중에 프로세스 1이 SUM에 접근하면 그것은 페지실패를 만나며 TreadMarks가 갱신된 페지에 이르게 한다.

다중쓰기

lazy해제일치성은 통보문흐름을 크게 줄일수 있다. 그러나 그것은 거짓공유문제를 해결하지 못한다. TreadMarks에서 거짓공유문제는 다중쓰기기술에 의하여 처리된다.

다중쓰기규약은 여러 프로세스들이 동시에 같은 페지에 대한 쓰기가능복사를 가지게 하여 프로세스들이 같은 페지에 대한 복사들에 동시에 쓸수 있게 한다. 물론 이러한 여

러 쓰기들을 조화시키는 방법이 있어야 한다.

TreadMarks다중쓰기규약은 아래에 서술한다.

2개의 프로쎄스 S1과 S2이 있어서 이것들이 같은 공유페지 P의 서로 다른 위치에 동시에 쓸것을 요구한다고 하자. S2이 P에 쓰면 TreadMarks는 P의 복사 P'(P의 쌍이라고 한다.)를 창조하고 P를 S1에 보존하며 P에 대한 쓰기보호를 해제한다. 다음 S1은 동기화(실례로 장벽)가 일어 날 때까지 보통의 국부기억과 같이 P에 여러번 쓸수 있다. 장벽에서 수정된 P는 diff라고 부르는 차분실행길이부호화를 창조하기 위하여 그것의 쌍 P'에 보존된 본래의 값과 비교된다. 다음 쌍 P'는 버려 지며 P는 무효된다.

S1이 후에 P에 접근하면 페지실패가 일어 난다. S2에 의해 만들어 진 diff는 S1에 보내지며 S1에서 diff는 페지 P에 적용된다.

10. 7. 참고문헌주해와 련습문제

Pfister[497], [498], Acderson et al.[33], Xu와 Hwang[657]은 다른 대변자들속에서 클라스터들에 대한 실례들을 모두 론하였다. POWER CHALLENGEarray는 SGI[548]에서 서술된다. SPARCCenter는 Cekleov et al.[134]에서 SPARCcluster는 [399]에서 서술된다. Sun Network FileSystem은 Sandberg et al.[527]에서 주어 진다. Compaq Recover Server Option Kit는 [164]에서 서술된다. Smith는 [446]에서 Microsoft Wolfpack를 서술한다. Tandem Himalaya는 [605]에서 서술된다. 보충적인 상업클라스터체계들에 대한 참조는 표 10-1에서 인용된다.

NOW과제는 [469]와 Auderson et al.[33]에서 서술된다. 봉사기 없는 xFS체계는 [34]에서, GLUnix조작체계는 [36]에서, 부하평가는 Arpaci et al.에서 서술된다. SHRIMP과제는 [237]에서 서술된다. Wind Tunnel은 [516]에서 서술된다. Syracnse WWVM과제는 [207]에서 서술된다. HKU Pearl은 부분적으로 [328]에서 서술된다. 클라스터연구과제들에 대한 보충적인 참고는 표 10-2에 인용된다.

클라스터에서 일관성공유기억기를 어떻게 지원하는가는 Reihardt et al.에서 론의된다. 메타계산의 개념은 Catlett와 Snarr[133]에서 처음으로 론의되였다. 클라스터프로그람작성을 위한 Starcluster는 Cabillic와 Puant[119]에서 서술된다. 광대역콤퓨터리용의 문제와 기술, 체계들은 Foster et al.[248, 249]와 Grimshaw et al.[282, 410], Dincer와 Fox[207], Fox et al.[251, 252]에서 론의된다.

IBM SP2의 구조는 a special issue of IBM System Journal(Vol. 34, N. 2, 1995)에서 서술된다.

POWER2처리장치는 Agarwal, et al.[13]에서, SP2구조는 Agerwala et al.[14]에서 취급된다. Cornell SP2구성은 [168]에서 서술된다. MHPCC SP2은 [445]에서 서술된다. SP2에서 사용된 Load Leveler는 [336]에서 서술된다. SP2의 통신쏘프트웨어는 Snir et al.[574]에서 주어 진다. SP2고성능교환기는 Stunkel et al.[592]에서 서술된다.

Digital의 AlphaServer8400은 Fenwick et al.[239]에서 서술된다. Digital Unix클라스터체계들에 대한 개괄은 Cardoza et al.[126]에서 주어 진다. VAXcluster는Shah[551]에서 서술된다. TruCluster는 Caroza et al.[127]과 Lawton et al.[397]에서 서술된다. Unix클라

스터의 우월성은 [194]에서, NT클라스터는 [195]에서 서술된다.

TruCluster의 TPC-C성능은 Piantenosi et al.[501]에서 주어 진다. 클라스터콤퓨터체계들에 대한 정보를 찾는 가장 좋은 장소는 제작업체들과 연구쎈터들의 매개 Web싸이트들에 있다. TreadMarks는 Amza et al.[30]과 Montotharassis et al.[373]에서 서술된다. 여기서는 또한 다른 쏘프트웨어 DSM체계들과 비교된다.

문 제

문제 10.1. Wolfpack의 유일한 특징들 즉 이 장에서 론의한 다른 클라스터들에서 찾을수 없는 특징들을 3가지 드시오.

문제 10.2. 3개의 유용성클라스터구성 즉 즉시대응와 활성이어받기, 실패허용클라스터들사이의 구조적 및 기능적차이들을 설명하시오. 매 유용성클라스터구성에 대하여 상업클라스터체계의 실례 2가지를 드시오. 상업응용들에서 그것들의 상대적인 장점과 약점을 론하시오.

문제 10.3. 10.3을 참고하고 IBM SP클라스터 /MPP체계에 대한 다음의 물음에 대답하시오.

(1) SP설계집단이 시장출하시간을 줄이기 위하여 내린 5가지 결정들을 설명하시오.
(2) SP를 일반목적체계로 되게 하는 특징 3가지를 드시오.
(3) SP가 어떻게 4가지 SSI특징 즉 단일입력자료와 단일화계층, 단일조종점, 단일일감관리체계를 지원하는지 설명하시오.
(4) 성능개선을 돕는 SP2의 구조적특징을 5가지 드시오.
(5) SP2통신부분체계에서 대역너비를 늘이기 위하여 사용된 기본기술들을 간단히 설명하시오.

문제 10.4. 10.4을 참고하고 Digital TurCluster에 대한 다음의 물음들에 대답하시오.

(1) DEC Tur클라스터는 공유없는 체계인가 아니면 디스크공유체계인가?
(2) 단일체계영상에 대한 다음의 국면들중에서 TurCluster가 어느것을 지원하며 어떻게 지원하는가를 설명하시오. 단일입력자료, 단일파일계층, 단일조종점, 단일기억공간, 단일프로쎄스공간.
(3) 확대가능성을 개선하는데 사용되는 TruCluster의 구조적특징 4가지를 드시오.
(4) TruCluster통신부분체계에서 통신부가처리를 줄이기 위하여 사용된 3가지 기술들을 간단히 설명하시오.
(5) TruCluster가 왜 일반통보문넘기기층(Universal Message Passing layer)을 받아 들이는가?

문제 10.5. 10.5을 참고하고 Berkeley NOW과제에 대한 다음의 물음들에 대답하시오.

(1) 단일체계영상에 대한 다음의 측면들중에서 어느것을 Berkeley NOW가 리용하며 어떻게 지원하는가를 설명하시오. 단일입력자료, 단일화계층, 단일조종점, 단일기억공간, 단일프로쎄스공간.
(2) 성능개선에 사용되는 Berkeley NOW의 구조적특징 4가지를 설명하시오.
(3) Berkeley NOW와 SP클라스터사이의 4가지 구조적차이를 설명하시오. 각각의 우월성에 대하여 론의하시오.

문제 10.6. Berkeley NOW에서 서술된 xFS를 깊이 생각하시오.

(1) 협조적인 파일캐쉬란 무엇이며 그 우월성은 무엇인가를 간단히 설명하시오.
(2) 어떤 조건에서 xFS가 협조적인 파일캐쉬에 효과가 있는가?
(3) 어떤 조건에서 xFS가 좋은 성능을 거둘수 없는가?

문제 10.7. Berkeley NOW에서 사용된 능동통보문규약을 깊이 생각하시오.

(1) 능동통보문들과 TruCluster통신규약사이의 3가지 차이를 드시오. 각각의 우월성에 대하여 설명하시오.
(2) 그림 10−20에서 왜 프로쎄스 P가 am−poll함수실행을 요구하는가?

문제 10.8. xFS를 깊이 생각하고 다음의 문제들에 대답하시오.

(1) xFS와 집중파일체계사이의 2가지 차이점을 설명하고 각각의 우월성을 론의하시오.
(2) 유용성을 높이기 위하여 xFS에 사용된 기본기술을 설명하시오.
(3) 작은 쓰기문제를 완화시키기 위하여 xFS에 사용된 기본기술을 설명하시오.

문제 10.9. 2개의 프로쎄스가 그림 10−26의 Jacobi오락프로그람을 실행하는데 사용된다고 하자. 여기서 float는 4바이트이며 페지크기는 4KB이다.

(1) 왜 3개의 장벽(barrier)이 있는지 설명하시오. 장벽들 매개가 지워 지면 어떤 일이 생기는가?
(2) 거짓공유가 있게 되는 필요충분조건을 유도하시오.
(3) 다중쓰기규약이 거짓공유효과를 줄이기 위하여 어떻게 사용될수 있는가를 보여 주시오.

문제 10.10. 2개의 프로쎄스 P0과 P1이 그림 10−27의 TreadMarks프로그람을 실행하는데 사용된다고 하자. Lazy해제일치성규약을 받아 들여 마지막 3개 행의 림계구역(critical region)코드가 어떻게 실행되는가를 자세히 설명하시오.

제11장. MPP 구성방식과 성능

기본적인 MPP문제들을 고찰한 다음에 이 장에서는 대규모의 병렬성을 지향한 두가지 서로 다른 수법들을 표현하는 현재의 MPP체계를 서술한다.

이 체계들은 NCC-NUMA수법을 표현하는 Cray T3E와 NORMA구성을 리용한 Intel/Sandia ASCI option Red들이다.

여기서는 에네르기성과 예약한 세가지 ASCI/MPP가동환경에 적용된 확대가능한 설계방책을 서술한다.

이 내용들은 세대교체를 전후하여 리용되게 될 다음세대의 MPP를 개괄한다.

체계구성외에 최근 년간 여러 독립적인 연구그룹들이 발표한 MPP성능평가실험에 대한 결과들을 고찰한다.

NAS결과들은 NASA Ames연구중심과 중국의 지능콤퓨터체계를 위한 국가연구중심에서 진행된 MPP성능평가실험에 기초한다.

MPI와 STAP결과들은 싸우스켈리포니아종합대학과 홍콩종합대학이 공동으로 진행한 MIT/STAP성능평가실험에 기초한다.

11.1. MPP 기술의 개괄

MPP(초대규모병렬처리)라는 용어는 그 의미가 시기에 따라 변화되였으므로 지난 시기에는 애매하게 사용되여 왔다.

현재 기술에 기초하면 그것은 수십만개의 처리소자로 이루어 진 대규모콤퓨터체계를 의미한다. 1997년까지 제작된 가장 큰 규모의 MPP구성은 Intel/Sandia option Red에서 9216개의 처리기를 가지고 있다.

11.1.1. MPP의 특성과 문제점

현재 MPP체계의 공통적인 구성방식을 그림 11-1에 제시하였다. 모든 MPP들은 물리

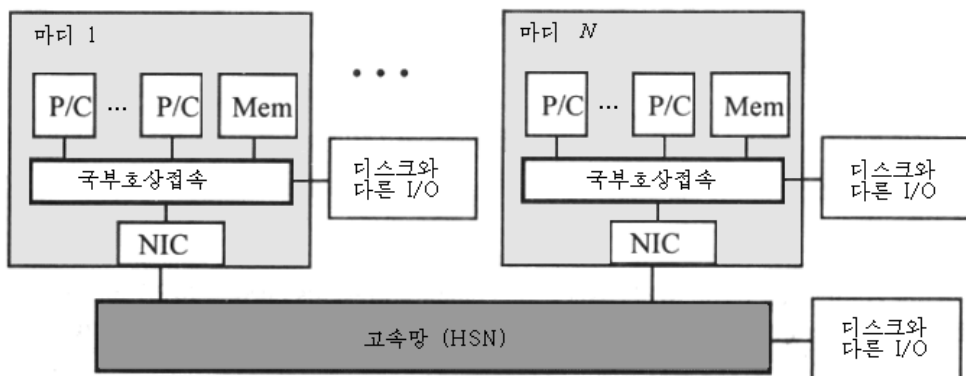



그림 11-1. 대규모병렬처리기의 일반적구성방식

적으로 분포된 기억을 사용하며 더욱더 많은 MPP들은 분포된 입출력(I/O)들을 사용하고 있다. 매 마디는 한개이상의 처리소자들과 캐쉬 (P/C)들, 국부기억, 그리고 한개이상의 디스크들을 가진다. 한개의 마디에는 처리기, 기억기, 입출력장치들을 련결하는 국부련결이 있다.

초기 MPP들에서는 이 국부련결이 보통 모선이였는데 최근 MPP들은 보다 높은 대역의 입력을 형절환회로망에 사용하고 있다.

매 마디는 망결합부회로(NIC)를 통하여 망에 련결된다.

확대가능성

MPP의 고유한 특징은 그것들이 수천개의 처리기에 이르기까지 확대할수 있게 설계된것이다. MPP들은 확대가능성을 증대시키기 위하여 다음과 같은 기술을 적용하였다.

- 물리적으로 분포된 기억구성방식을 사용한다. 이것은 집중형기억구성방식보다 높은 집합기억폭을 제공하며 따라서 확대가능성을 보다 높여 준다.
- 기억기와 입출력장치능력사이에 균형적으로 처리된다. 비례적으로 빠른 기억기와 입출력부분체계가 없이는 자료를 처리소자에 충분한 속도로 전송할수 없으므로 사실상 빠른 처리기는 전혀 가치가 없다.
- 병렬성과 호상작용능력사이의 균형계산능력 다시 말하여 <프로쎄스>/<스레드>관리, 통신, 그리고 동기화로 인한 무효시간이 실행시간을 관리할수 있다.

체계원가

MPP에는 많은 마디와 호상련결요소들이 필요하므로 요소당 원가를 낮추어야 한다. 원가를 낮추는 여러가지 수법들이 리용되고 있다.

- 현재의 상품화된 CMOS극소형처리기를 리용한다. 이 처리기들은 원래 워크스테이션과 봉사기를 위하여 개발되였다. 그 가치는 상업적특성의 비용이 낮고 투자를 늘일수 있게 하는것이다. 이것은 그 처리소자의 성능을 18개월부터 24개월까지를 주기로 하여 성능을 2배로 늘인다(Moore법칙).
- 세대별 확대가능한 안정한 구성방식을 리용한다. 이러한 한가지 기술이 1.1.3에서 론의한 대면부구성방식이다.
- 물리적으로 분포된 기억기구성방식을 리용한다. 이것은 같은 기계적크기를 가지는 집중형기억기구성보다는 비용이 낮다.
- SMP마디들을 리용한다. 이것은 호상련결된 규모를 축소시킨다.

실례 11.1. TMCC−2 소편전용설계로부터 얻은 교훈

지난 시기에 MPP들을 제작하기 위한 실천에서는 수백수천의 전용방식으로 설계된 “단순한”마디들의 호상련결이였다. 실례로 사유하는 기계를 들수 있다. 여기에서 매 마디들은 본질상 확장된 산수연산부(ALU)등록기들과 작은 국부기억기이다.

간단히 하기 위하여 여러 마디들을 한개 소편으로 집적시킬수 있다. 이리하여 마디

당 비용이 낮아 졌다.

그러나 개별적인 마디들은 매우 낮은 성능을 가진다.

상품화된 한개의 마이크로처리소자의 성능에 조화되도록 하려면 그런 마디 열두개가 필요하다. 그것들은 전용설계되였으므로 Moore법칙에 따르지 않는다.

실례 11.2. 세대별 확대가능성을 보장하는 대면부연구방법

안정한 구성방식의 실례로 이 장에서 론의하는 모든 MPP들은 어느 정도 대면부연구방법을 따른다(그림 1.2).

마이크로처리소자가 다음세대로 발전함에 따라 체계의 나머지부분은 변화되지 말아야 한다. 더우기 MPP들모두는 일정한 마디등급을 가진다. 실례로 Cray T3D/T3E에 대하여 등급은 6(3D−torus)이며 Intel Paragon에 대하여서는 4(2D−mesh), 그리고 IBM SP2(다단계오메가(multistage omega)망)에 대하여서는 1이다.

현재 상품화된 마이크로처리소자들은 PC, 워크스테이션, 그리고 SMP봉사기(MPP를 위한것이 아닌)들과 같은 그런 작은 체계들이다.

마이크로처리소자들을 리용하면 많은 확대가능성과 비용−효과를 얻는 반면에 DSM 구성방식에 기초한 MPP에 대한 여러가지 문제점들을 발생시킨다.

MPP설계자가 고찰하여야 할 문제목록의 일부는 다음과 같다.

- 마이크로처리소자들은 충분히 큰 물리적주소공간을 가지지 않는다. 실례로, Alpha 21064마이크로처리소자는 Cray T3D MPP에서 리용된다. 이 처리소자는 다만 2^{33}개 바이트(8−기가바이트)의 물리적주소공간을 차지한다. 반면에 T3D는 최대로 128기가바이트(GB)의 물리적기억을 가진다. Cray T3D설계가들은 물리적주소공간의 규모를 확장하기 위하여 DTB Annex라는 특수한 하드웨어를 첨가하였다.
- 마이크로처리소자는 충분히 큰 TLB를 가질수 없으며 TLB비적중은 캐쉬비적중보다 훨씬 더 값이 비싸다. 대규모자료모임과 불규칙적인 기억접근패턴의 리용에 대하여 TLB비적중들은 성능을 크게 감소시킬수 있다.
- 마이크로처리소자들은 한번에 한개의 캐쉬행의 기억에 접근한다. 이것은 한 단어씩의 접근을 무효로 한다. 현재 마이크로처리소자는 오직 작은 척도로 비블로크화된 캐쉬만을 지원하며 한두개의 우월한 기억기참조를 허용한다. 이것은 MPP에서 요구되는 지연허용능력을 제한한다.
- 그 계산능력에 비하면 마이크로처리소자는 불충분한 조작체계자원을 가진다. 례외적인 처리와 교차보호경계는 값이 비싸다. 이것은 공정관리, 통신 및 동기화의 효과적인 지원을 어렵게 한다. 이 문제는 7장에서 론의하였다.

일반성과 사용가능성

현세대 MPP들은 지난 시기 경험으로부터 학습한다.

다시 말하여 MPP가 성공적인것으로 되자면 그것이 서로 다른 응용(례하면 기술적이

든 상업적이든) 그리고 서로 다른 알고리듬방식, 서로 다른 조작방식을 지원하는 그런 목적지향적인 체계로 되여야 한다.

여기서는 독특한 구성방식상 비정상적인 주위환경의 리용에 중점을 두면서 적합성이 적은 응용은 지원하지 말아야 한다.

보다 명백히 말하면 현재의 MPP들은 다음과 같은 특징들을 가진다.

- MPP들은 비동기MIMD방식을 지원한다. SIMD는 일반목적MPP들에서 제외된다.
- MPP들은 통보문넘기기(PVM와 MPI) 그리고 자료병렬(HPF)과 같이 널리 보급된 표준프로그람모형들을 지원한다.
- 마디들은 여러 분할구획들로 배렬되며 호상작용하는 묶음방식으로 크고작은 작업들을 지원한다.
- 서로 련결된 위상은 자유롭게 련결된 마디들로 이루어 진 모임을 보는 사용자로부터 은페되여 있다.
- MPP들이 서로 다른 준위에서 단일체계영상을 지원한다. 밀접하게 결합된 MPP들이 흔히 분포된 조작체계를 리용하며 하드웨어준위와 OS준위에서 단위체계형태를 제공한다.
- 이것은 1000개의 처리기 MPP들가운데 적어도 한개의 처리소자가 매일 고장날것이라고 생각된다[320]. MPP들은 높은 리용기술을 사용하여야 한다.

통신요구

MPP들과 현재의 워크스테이션묶음(COW)들사이의 기본차이점은 마디사이 통신에 있다. COW에서 마디들이 때로는 표준LAN을 통하여 련결된다. MPP에서 마디들은 높은 속도, 그리고 최대의 대역너비와 작은 지연을 가지는 적중한 망에 의하여 호상련결되여 있다. 더우기 최대성능을 실현하기 위하여 적중한 통신쏘프트웨어가 보장되여 있다.

이 모든것은 현재의 MPP들이 통신성능상 COW들보다 우월하다는것을 의미한다. 그러나 앞으로 10년안에 표준망기술들에서 급속한 전진이 이룩되리라는것을 예견할수 있다.

MPP에서 이 좋은 통신지원을 말은 가지가 앞으로 COW에서 리용되는것보다 얼마나 더 오래 유지될수 있겠는가는 말하기 어렵다.

기억기와 I/O능력

이것들은 큰 규모로 확대될수 있기때문에 MPP들은 다른 구성방식에서 찾아 볼수 없는 대규모집약기억과 디스크용량을 보장할수 있다.

그밖에 상품화된 MPP들은 고속I/O체계에 특별한 주의를 돌려야 한다.

많은 체계들에서는 기억기뿐아니라 I/O부분체계도 물리적으로 분포되여 있다.

그러나 I/O의 발전은 체계의 나머지부분에 비하여 더디다. 그리고 확대가능한 I/O부분체계를 보장하는 방법은 적극적인 연구분야의 하나이다. 참고문헌들과 Web자원은 고성능 I/O연구대상들에 대한 지적자들을 포함하고 있다.

11. 1. 2. MPP체계들에 대한 개괄

표 11-1은 큰 구조의 체계들에 대한 세가지 서로 다른 방식을 표현하는 세가지 현대적인 MPP들의 구성방식적특징들을 렬거하고 있다.

IBM SP2는 10.3에서 론의한바와 같이 MPP의 제작에 대한 묶음양식을 제공한다.

Intel ASCI체계는 작은 마디들 및 견고하게 결합된 회로망호상련결, 계산마디우에서의 극소핵심부조작체계를 가지는 보다 전통적인 MPP수법을 따른다.

Intel Paragon MPP체계의 Intel ASCI체계는 Intel Paragon MPP체계를 계승하고 있다.

SP2과 Intel ASCI체계는 둘다 NORMA모형을 가지는 통보문넘기기용다중콤퓨터이다.

마디점사이의 통신은 NORMA기계에서 넘기기되는 명백한 통보문에 의존한다.

SGI/Gray Origin 2000은 대역적으로 접근가능하며 물리적으로 분포되여 있는 캐쉬응집을 위한 하드웨어지원을 가지며 기억체계에 의하여 특징 지어 지는 MPP구성에 대한 서로 다른 방식을 표현한다.

류사한 CC-NUMA구성방식을 리용한 다른 MPP는 Hp/Convex Exemplar X-Class이다.

Cray T3E체계도 역시 DSM기계이기는 하지만 캐쉬응집을 위한 하드웨어지원은 없다. 따라서 이것은 NCC-NUMA기계이다.

이러한 DGM기계의 고유한 프로그람작성환경은 공유변수모형을 보장한다.

표 11-1　　세가지 초병렬처리기(MPP)들의 비교

MPP마디	Intel/Sandia ASCI Option Red	IBM SP2	SGI/Cray Origin 2000
큰 표본구성	9072처리기, 1.8Tflop/s at SNL	400처리기, 100 Gflop/s at MHPCC	128처리기, 51 Gflop/s at NCSA
리용가능한 실지날자	1996년 12월	1994년 12월	1996년 10월
처리기형태	200MHz, 200Mflop/s Pentium Pro	67 MHz, 267 Mflop/s POWER2	200MHz, 400 Mflop/s MIPS R1000
마디구성방식과 자료기억	2개 처리기 32-256MB기억, 공유디스크	1개 처리기 64MB-2GB국부기억, 1-4, 5GB국부디스크	2개 처리기 32MB-256GB DSM, 공유디스크
호상접속과 기억기모형	분산2Dmesh, NORMA	다단계망 NORMA	굵은 하이퍼립방체 CC-NUMA
마디연산체계	가볍게 무게화된 핵심부 (LWK)	Complete AIX (IBM Unix)	극소핵심부 Cellular IRLX
고유한 프로그람작성기구	PUMA Portals에 기초한 MPI	MPI와 PVM	Power C Power Fortran
다른 프로그람 작성기구	Nx, PVM, HPF	HPF, Linda	MPI, PVM

응용프로그람작성준위에서 모든 MPP들은 C언어, 포트란언어, HPF, PVM과 MDI와 같은 현재의 지원표준언어와 서고들이다.

과거 MPP들

지난 시기 MPP들은 과학적인 초고속계산에 1차적으로 리용되였다. 주목할만한 체계들은 사유하는 기계인 CM2/CM5, NASA/Goodyear인 MPP, nCUBE, Cray T3D/T3E, Intel Paragon, Maspar MPI, Fuiiesu Vpp500 그리고 KSR1 등등을 포괄하고 있다.

이 MPP들가운데서 어떤것은 벡토르하드웨어를 가지든지 혹은 SIMD 섬세한 자료병렬성만 가진다.

초기 MPP의 일부가 1993년에 출판된 Hwang교수가 쓴 《Advanced Computer Architecture》[327]에서 취급되였다.

오늘 이 기계들의 대부분은 없어 졌다. 그리하여 많은 사람들은 다른 많은 초고속콤퓨터회사들가운데서 사유하는 기계회사(Corporation), C ray Research Inc, Intel Systems Division들의 역할이 낮아 짐과 함께 MPP들이 사라졌다고 생각하고 있다.

현실은 최근 년간에 병렬처리가 공업용, 상업용, 업무용응용프로그람에 대한 요구가 높아 짐과 함께 갱신되고 있다.

상업용MPP응용프로그람들

MPP들에 대한 대부분의 기술문헌들과 연구자료들이 기술공학적계산에 집중되고 있다. 많은 연구론문들이 MPP우에서 병렬처리가 큰 회피문제들을 얼마나 풀수 있는가를 론의한다.

이것들은 MPP가 매우 거대한 병렬적인 과학기술계산응용에만 좋다는 그릇된 인상을 줄수 있다.

사실상 많은 MPP들은 상업용응용프로그람 및 망응용프로그람들에서 성과적으로 리용되고 있다. 례컨대 1997년까지 판매된 3000개의 SP2체계가운데서 약 절반이 상업적응용프로그람에 적용되였다. 나머지 절반가운데서 큰 몫이 LAN공고화에 적용되고 있으며 작은 몫에 해당한것만이 과학적고속계산에 리용된다.

상업적MPP응용프로그람의 특별히 가장 널리 응용되는 분야의 하나는 자료기지, 결심지원체계, 그리고 수자식서고들이다.

확대가능성, 리용가능성 및 관리가능성은 상업적고성능응용프로그람시장에서 더욱 중요하다.

11.2에서 Cray T3E/ MPP를 학습한 다음 우리는 11.3에서 MPP의 완전히 새로운 세대를 고찰할것이며 ASCI option Red체계의 일부 구체적인 내용은 11.4에서 취급한다.

11. 2. CrayT3E 체계

1995년에 개발된 Cray T3E는 1993년에 생산된 Cray T3D체계의 계속이다.

이것은 보다 빠른 요소들을 리용하며 여러가지 구성방식적변화들을 받아 들여 T3D보다 성능을 개선하였다.

이것들은 T3D구성방식의 응용환경과 성능평가실험렬로부터 이루어 진다[40, 361, 340].

11. 2. 1. T3E의 체계구성방식

그림 11-2에서 보다싶이 Cray T3E는 분산공유기억다중처리소자이다. 이 체계는 고속통신을 보장하기 위한 3D쌍방향원환체망과 I/O장치에 의한 련결을 보장하는 여러개의 기가링(GigaRing)통로에 의하여 호상련결된 여러개의 처리요소(processing element-PE)로 이루어 져 있다.

T3E의 구성방식속성은 표 11-2에 목록으로 제시하였다.

T3E-900은 1996년 말에 발표된 T3E의 개량형이다.

그 기본차이점은 T3E-900이 고속처리소자박자(450MHz T3E에서 300MHz이지만)를 리용하여 낮은 비용을 제공해 주는것이다(T3E에 대하여 1M$이지만 T3E-900은 500K$보다 낮은것에서 시작된다).

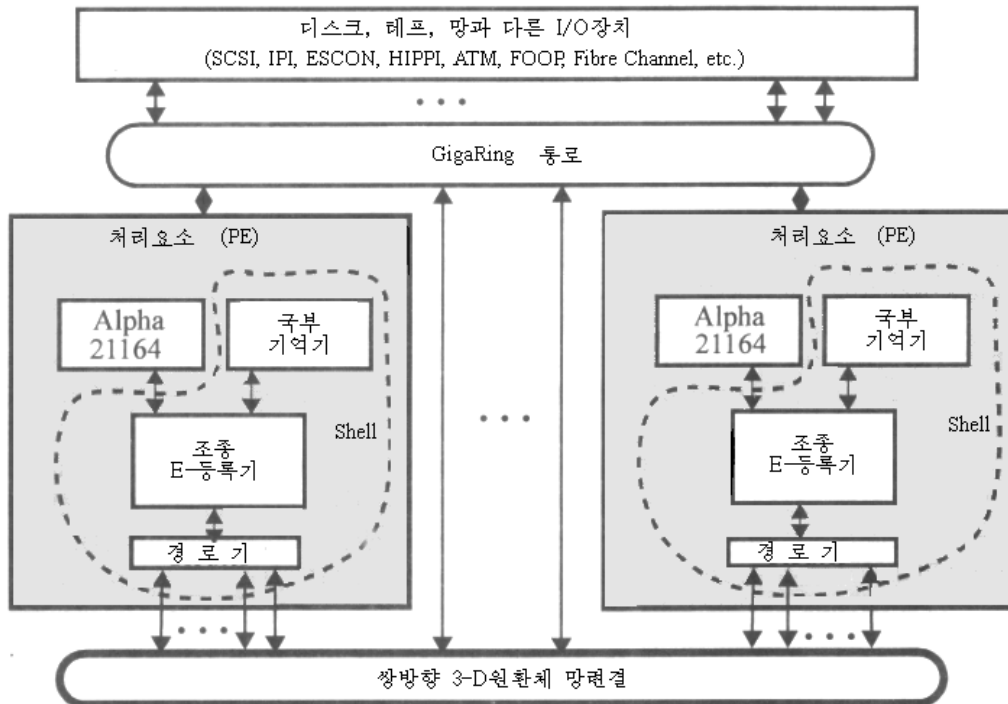



그림 11-2. Cray T3E 구성방식도표

처리요소

T3E에서 매 PE는 대면부회로에 의하여 구성된 DEC Alpha 21164극소형처리소자로 구성되여 있다.

대면부회로는 국부기억, 조종소편, 경로기소편들을 포함한다. 대면부체계론리는 75MHz에서 실행된다.

21164처리소자는 최대속도 600Mflop/s를 가지고 300MHz에서 박자화되여 있다.

국부기억은 64-GB까지의 용량을 보장하며 최대 1.2GB/s의 대역을 보장한다(470MB/s에 달하는 STREAM Copy benchmark).

경로조종기소편은 7개의 쌍방향포구들을 가지는데 하나는 PE에 련결하기 위한것이며 나머지 6개는 3D원환체망말단에 련결하기 위한것이다.

주문형식으로 만들어 진 조종소편은 PE의 모든 국부기억들로 이루어 진 분산공유기억을 실현한다.

모든 처리소자들은 임의의 PE에 있는 기억에 접근할수 있으며 매 PE는 기가링통로를 통하여 임의의 I/O장치들에 접근할수 있다. 이 소편은 또한 지연은페와 효과적인 동기화지원을 담당할수 있다.

표 11-2 Cray T3E 와 T3E-900 들의 성능속성

속성	T3E	T3E-900
처리기박자주파수(MHz)	300	450
처리기최대속도(Mflop/s)	600	900
처리기개수	6-2048	6-2048
체계최대속도(Gflop/s)	3.6-1228	5.4-1843
물리적기억용량(GB)	1-4096	1-4096
총 최대기억대역(GB/s)	7.2-2450	7.2-2450
I/O의 최대통로개수	1-128	1-128
총 I/O 최대대역(GB/s)	1-128	1-128
3-D torus최대련결대역(MB/s)	600	600

T3E PE들은 기판준위의 캐쉬를 가지지 않는 대신에 21164처리소자에 있는 캐쉬를 리용한다.

캐쉬가 붙어 있는 소편은 두가지 준위가 있는바 그 첫 준위는 8-KB의 명령캐쉬와 8-KB의 자료캐쉬로 이루어 져 있다. 둘째 준위는 련결된 3-통로모임인데 명령과 자료에 통일적으로 리용할수 있는 90-KM의 캐쉬이다.

기판준위의 캐쉬를 리용하지 않고 주기억대역을 개선할수 있다.

호상련결

T3E는 총 6개의 매 방향에서 단위체계박자(13.3ns)의 64-bit단어를 넘기기할수 있는 3D원환체망[541]을 통하여 낮은 지연과 높은 전역통신을 지원한다.

521-PE체계에 대한 쌍대역은 122GB/s를 넘는다.
망은 통보들이 열점들을 선회할수 있도록 적응적인 최단경로알고리듬을 실행한다.

I/O부분체계

T3E I/O부분체계의 중심에는 원환체망에도, PE에도 련결되는 기가링통로들이 있다. 매 기가링통로는 높은 대역과 믿음성을 보장하기 위하여 서로 반대방향으로 오고 가는 두개의 고리안에 자료를 가지는 거꿀순환 32bit 쌍이다.

기가링통로에 16개까지의 PE들이 련결될수 있으며 또한 매 통로는 최대로 1GB/s의 대역을 넘기기할수 있다. 최대의 구성체계에서 다중기가링통로를 가진 I/O부분체계는 128GB/s의 I/O대역을 보장할수 있다.

PE외에 다음과 같은것들을 포함하는 기가링통로에 여러가지 형의 I/O마디들이 련결되여 있다.

- 다목적마디. 이것은 SCSI, FDDI, 이써네트
- 디스크마디. 이것은 Fibre통로와 IPI들 그리고 ATM들을 보장하도록 Sbus조종카드들을 받아 들인다. 디스크와 RAID를 지원한다. 또한 테프마디 Block Mux와 ESCON테프구동기들을 지원한다.
- HIPPI마디, 100MB/s 혹은 200MB/s HIPPI통로들을 지원한다.

１１．２．２．T3E의 체계쏘프트웨어

Cray C 90을 앞단으로 요구하는 T3D와는 달리 T3E는 self-host체계이다. T3E는 핵심부준위에서 단일체계영상을 보장하는 분산형조작체계인 UNICOS/mk라고 불리우는 Cray 64-bit유닉스체계(UNICOS)를 이끌어 낸다.

T3E는 공유변수, 통보문과 그리고 자료병렬프로그람작성을 지원하는 종합적인 환경을 보장한다.

조작체계

UNICOS/mkUNICOS/mk체계는 국부적 및 대역적봉사기들로 나누어 진다. PE는 사용자와 체계PE 로 나누인다. 사용자PE 는 사용자응용프로그람과 지령을 실행한다. 체계PE는 대역적조작체계봉사를 제공한다.

매 사용자PE는 국부봉사기와 CHORUSCHORUS체계[156]로부터 유도된 유닉스극소핵심부를 포함하고 있다.

모든 처리과정특유의 요구들은 극소핵심부와 기억배치뿐아니라 통보문/자료의 넘기기도 포함하는 국부봉사기에 의하여 처리된다. 국부봉사기들은 파제관리, 파일공간배치, 계획화, 안정성 그리고 I/O관리들과 같은 그러한 체계규모봉사를 보장한다.

UNICOS/mk체계는 검사점/재기동, 파일공유체계, 담보된 파일전송 등이 지원되는 핵심부를 거쳐 자동적인 작업회복을 지원한다. 이것은 자원관리, 업무관리, 체계감시, 회계, 작업계획화 그리고 기밀보호봉사기를 보장하는 한조의 도구를 포함하고 있다.

확대가능한 I/O를 실현하기 위하여 UNICOS/mk체계는 분산형파일체계관리를 실행한다. 사용자 PE의 국부파일봉사기술은 완충기없이 <읽기>/<쓰기>요구를 봉사한다.

대역파일봉사기들은 파일열기 및 닫기와 같은 적게 쓰이는 요구에 대하여서만 리용하여야 한다.

다중파일봉사기들은 병렬 I/O전송을 진행할수 있도록 한다.

프로그람작성환경

T3E는 포트란 90, C, 그리고 C++, 최량화되고 병렬성된 과학적 및 수학적서고들에 대한 최량콤파일러를 보장한다.

T3E는 포트란 90과 HPF언어를 리용한 자료병렬프로그람작성, PVM과 MPI서고들에 대한 통보문절달형프로그람작성, Cray공유기억서고 SHMEM와 CRAFT콤파일러지령과 서고부분프로그람루틴을 리용한 변수공유프로그람작성을 지원한다.

이 도구들을 임의로 결합하여서도 리용할수 있다. 그밖에도 T3E는 다음과 같은것을 포함하는 효과적인 병렬프로그람의 개발을 도울수 있도록 환경도구모임을 보장한다.

- Total View라고 불리우는 병렬응용프로그람을 위한 기초적인 원천준위의 오유수정프로그람. 이것은 사용자로 하여금 개별적인 과정 혹은 과정그룹의 실행행정을 조종하며 현시할수 있도록 한다.
- MPP Appentice라고 불리우는 성능병렬분석도구. 이 도구는 성능정보를 해석하며 성능개선을 권고하는 전문가체계를 특징 짓는다.

11. 3. 새 세대 ASCI/MPP

현재 고성능초고속콤퓨터개발을 위하여 두가지 중요한 노력을 기울이고 있다. 다시 말하여 그것은 petaflops project와 ASCI프로그람이다.

petaflops project는 고속형콤퓨터의 속도를 Pflop/s(10^{15}flop/s)에 이르도록 하는것을 오래전부터 목표로 설정하였다.

최근의 연구결과는 합리적인 기술과 commodity off-the-shelf(COTs)요소를 리용하면 이러한 MPP가 2005년까지 10억딸라의 비용으로 제작될수 있다는것을 보여 주고 있다.

더 고급한 기술을 리용하면 2007년까지 1Pflop/s초대형콤퓨터가 제작될수도 있다(물론 비용은 예견할수 없다.).

이 후자는 콤퓨터공업의 기본흐름에 대한 희망을 담고 있으므로 새로운 기술이 COTS를 타승할수 있겠는가는 분명하지 않다.

어느 방법이든지간에 1Pflop/s체계는 100만이상의 스레드(thread)에 이르기까지 수천개처리기의 범위에서 동시에 초병렬성을 효과적으로 개발하였다. 독자들은 petaflops project에 대한 더 많은 정보를 알기 위하여 Web자원에 문의하여야 한다.

1994년에 에네르기성(DOE)은 ASCI(Accelerated Strategic Computing Initiative)프로그람을 개발하였다.

이 프로그람은 10년동안에 1억딸라를 지출하였는데 이것은 핵무기저장고의 로화효과, 생명공학, 의학, 제약연구, 기상예보, 비행선과 자동차설계, 기술공정개선, 환경보호 등을 모의하는 Tflop/s초대형콤퓨터체계를 구성하는데 리용되였다.

11. 3. 1. ASCI 확대가능체계의 설계전략

ASCI프로그람은 지하핵시험을 위한 대용물로써 초대형콤퓨터를 개발할것이다. ASCI 프로그람의 목표는 1996년까지 1Tflop/s체계, 2000년경에 10～30Tflop/s체계, 2004년까지는 100Tflop/s체계를 설계하는것이다.

이 체계들은 류사한 비용으로 개발리용될것이다.

1994년의 1Gflop/s로부터 2004년의 100Tflop/s까지에 도달하면 10년동안에 10^5배로 성능이 개선된다.

이것은 무어법칙이 존재하는 콤퓨터공업의 현 추세를 따른다면 도달하지 못할것이다. 무어법칙을 따르면 10^5배의 성능개선은 같은 비용이 보장되는 조건에서 20~32년이 걸릴것이다. 이것은 100Tflop/s의 성능은 2025년까지도 달성할수 없다는것을 암시하여 준다.

ASCI프로그람을 채용하는 방책은 두가지 주목할만한 특징을 가진다. 그 특징은 가속적인 개발과 균형적인 확대가능한 설계이다.

ASCI프로그람은 바로 최대속도가 아니라 응용프로그람성능이 1994년의것보다 5급의 더 높은 지속적인 응용프로그람성능을 부여할수 있도록 체계 총체를 해결하는것을 목적으로 한다.

이것은 다음과 같은 가속화된 단계들로 특징 지을수 있는것으로서 균형적인 확대가능한 설계방법을 요구한다.

- 시장의 사탕매점과 같은것으로 생각하면서 대중장마당가동환경이나 응용프로그람을 대할것이 아니라 과학적계산응용프로그람의 높은급가동환경이라는데 주의를 돌린다.
- off−the−Shelf(COST)하드웨어와 쏘프트요소들을 될수록 많이 리용한다.
- 대규모병렬구성방식을 리용한다. 한개의 체계화상을 가진 유효한 가동환경에 무한개의 COTS마디들이 배렬될수 있는 집적기술과 확대기술에 주의를 돌린다.

확대가능한 설계도

표 11−3은 균형적으로 확대가능한 계산환경을 보장하는 ASCI설계도를 제시하였다. 이것은 2003년까지 핵에네르기와 무기연구프로그람의 초대형계산요구에 부합된다.

표 11-3 MPP 들의 특징에 대한 균형적설계를 위한 ASCI 전략

속성	1996	1997	2000	2003
응용프로그람성능 (시간)	1		1000	100000
최대계산속도 (Gflop/s)	100	1000	10000	100000
기억용량(TB)	0.05	0.5	5	50
디스크용량(TB)	0.1-1	1-10	10-100	100-10000
문서보관용량(PB)	0.13	1.3	13	130
I/O속도(GB/s)	5	50	500	5000
망속도(GB/s)	0.13	1.3	13	130

균형적설계방책은 다음과 같은 공학적고찰을 고려하고 있다.

- 말단 대 말단성능(End-to-End Performance). 리상적인 최대속도 혹은 성능평가실험은 충분하지 못하다. 탁상콤퓨터앞에 앉아 있는 과학자는 목표프로그람의 성능이 10^5배로 개선되였다는것을 보게 될것이다. 이것은 코드개발, 작업분담 및 실행, 결과보기 등의 전과정을 포괄한다.
- 균형적이며 확대가능한 하드웨어. 하드웨어는 확대가능하고 균형적이여야 한다. 이것은 탁상콤퓨터와 망하드웨어로부터 초대형콤퓨터가동환경에 이르기까지 여러가지로 확대가능하다는것을 의미한다. 균형적설계규칙은 다음과 같이 설정된다. 즉 1Gflop/s의 최대속도는 1GB기억기, 50GB디스크, 10TB문서보관, 16GB/s캐쉬대역, 3GB/s기억기대역, 0.1GB/s I/O대역, 10MB/s디스크대역 그리고 1MB/s문서보관대역들과 정합될것이다.
- 균형적확대가능한 쏘프트웨어. ASCI는 10부터 100배까지의 성능개선이 새로운 쏘프트웨어가 개발됨으로써 달성될것이라는것을 담보한다. 이러한 쏘프트의 요구들은 과학자들이 초대형콤퓨터를 효과적으로 리용할수 있을뿐아니라 그 생산성을 개선할수 있게 한다. ASCI는 이러한 도구들이 없으면 생산성과 질이 높은 응용프로그람을 개발하는데 30년이상 걸릴것으로 예견한다.

11. 3. 2. 하드웨어와 쏘프트웨어에 대한 요구

ASCI기금은 ASCI표준가동환경을 제작하기 위하여 세개의 주요콤퓨터회사들에 하드웨어와 쏘프트웨어요구의 완전모임을 지시하였다.
이 요구들은 아래에서 다음과 같이 해석되여 있다.

하드웨어요구
처리기, 계층기억기성, I/O부분체계들에 대한 요구들은 ASCI초대형콤퓨터를 제작하

기 위하여 모두 규정되였다. 실례로 ASCI기억기요구들은 표 11-4에 제시되였다.

표 11-4 에네르기성에 의한 ASCI 기억기요구들

기억순위	유효지연시간 (CPU cycles)	읽기, 쓰기 대역너비*	기억용량**
소편내장 캐쉬, L1	2부터3	16부터 32B/cycle	10^{-4}
소편외장 캐쉬, L2	5부터6	16B/cycle	10^{-2}B/flop/s
국부주기억	**30부터 80 (30부터 50)**	**2부터 8B/flop최대 (2부터 8B/flop 유지)**	1B/flop/s
이웃마디들	**300부터 500 (30부터 50)**	**1부터8B/flop (8B/flop)**	1B/flop/s
멀리 떨어 진 마디들	**1000 (100부터 200)**	**1B/flo p**	1B/flop/s
I/O속도(기억기-디스크)	10ms	0.01부터0.1B/flop	10부터 100B/flop/s
문서(디스크-테프)	초당	**0.001B/flop (0.01부터0.1B/flop)**	**100B/flop/s (10^4B/flop/s)**
사용자접근시간	0.1s(1/60s)	**OC3/탁상형(OC12-48/탁상형)**	사용자 100명
다중싸이트(sites)	0.1s	알려 지지 않음	알려 지지 않음

주의 : 굵은 서체입력자료는 1997년의 공업수준에 대응하는것이며 가는 서체입력자료는 그렇지 않은것에 대응한다. 대부분의 입력자료는 1998년의 수준에 대응하는것이며 괄호안에 있는 입력자료들은 2000년에 해당한다.

* 단위작업부하당 대역너비 혹은 CPU주기당 대역너비. ** 단위속도(flop/s)당 용량

두가지 요구기록모임이 있는바 대부분의 입력기록들은 1998년을 지향한 요구들이다. 괄호안의 입력기록들의 2000년까지 예견한것들이다.

굵게 쓴 입력기록들은 공업수준이 1997년에 그 요구를 충족시킬수 없다는것을 의미하는것이며 가늘게 쓴 입력기록들은 현재의 공업수준에서 도달할수 있는것들이다.

국부기억기처리기당 요구들을 고찰한다.

처리기가 200MHz박자(CPU박자시간은 5ns)와 400Mflop/s의 속도를 가진다고 가정한다.

그러면 표 11-3에 기억기지연요구가 1998년까지는 150～400ns이며 2000년까지는 75～150ns에 해당하였다.

우연적인 <읽기>/<쓰기>를 위하여서는 기억기대역이 최대로 800부터 3200MB/s까지 (2000년가지는 동일하게 유지된다.) 이를것이다. 그리고 국부기억기능력은 최소한 400GB일것이다.

쏘프트웨어요구

ASCI쏘프트웨어요구는 표 11-5에 제시되였다. 쏘프트웨어령역에서 공업은 훨씬 뒤떨어 져 있다. 그러므로 기본노력이 쏘프트웨어개발에로 지향된다. <사람>/<콤퓨터>대면부, 응용프로그람과 프로그람작성환경, 분산조작체계 등에서의 ASCI의 요구사항은 아래와 같다.

1) **<사람>/<콤퓨터>대면부** 시각화와 인터네트기술.
2) **응용프로그람환경** 수학적알고리듬, mesh생성, 령역분할, 과학적인 자료관리.
3) **프로그람작성환경** 프로그람작성모형, 서고들, 콤퓨터들, 오유수정기들, 성능도구들, 대상기술.
4) **분산형조작쏘프트웨어** I/O, 파일 및 보관체계, 믿음성, 통신체계관리, 분산형자원관리
5) **진단성능감시기** 체계의 정상성감시

표 11-5 ASCII 쏘프트웨어요구와 공업적인 준비

쏘프트웨어	보안	확대가능성	기능성	이식가능성
인간/콤퓨터대면부	↑△	↓△	가시화↓△ 인터내트↑△	↑●
응용환경	↑●	↓△	↓△	
프로그람환경	↓△	↓△	↓△	↓△
분산연산쏘프트웨어	↓△	↓△	↓△	↓△
진단성능모니터	↑●	↓△	↑●	↓●
공업적요구충족 ; 공업요구불충족 ; 요구는 같다. ; 요구는 시간에 따라 증가 2중입구자료들은 동시에 결합요구를 가리킨다.				

11. 3. 3. 수축된 ASCI/MPP 가동환경

ASCI화에서 표 11-6에 제시된것처럼 세가지 계획이 시작되였다.

Intel은 1Tflop/s이상의 보장된 라인패크(Linpak)성능을 가지는 기계(코드이름은 option Red)를 제작하는데 5천 5백만딸라를 투자하였다. OptionRed기계에 대하여서는 11.4에서 론의하였다.

IBM은 Lawrence Livermore국립연구소에서 1998년에 3Tflop/s초대형콤퓨터(코드이름은 Blue Pacific)를 제작하여 직결시키는데 9천 300만딸라를 투자하였다.

분명히 IBM Deep Blue계획은 공동의 관심사로 되였다. Blue pacific option의 세부들은 1998년 말까지도 혹은 그이후에도 공개되지 않을것이다.

표 11-6 세가지 ASCII MPP 가동환경의 요약

특징	Intel/SNL Option Red	IBM.LLNL Blue Pacific	SGI/LANL Blue Mountain
처리기선택	200-MHz Pentium Pro with 200 Mflop/s	200-MHz POWER3 with 800 Mflop/s	1-Gflop/s MIPS processor
체계구성방식	NORMA-MPP	Cluster	CC-NUMA
처리기수	9216	4096	3072
최대속도	1.8Tflop/s	3.2Tflop/s	Over 3 Tflop/s
기억기용량	594GB	N/A	N/A
디스크용량	1TB	75TB	N/A
유효날자	1997.6	1998.12	1998.12

SGI/Cray는 두개의 초고속콤퓨터를 결합하여 4Tflop/s이상의 최대성능을 가지도록 한것으로써 그 제작을 위하여 1억 1천 50만딸라를 계약하였다.

3Tflop/s의 속도를 목표로 하는 3072개의 처리기로 구성된 한개의 체계(코드이름은 Blue Mountain)는 1998년 12월까지 Los Alamos국립연구소에 진행되도록 예견되여있었다.

ASCI/MPP option의 개발과 응용은 앞으로 주목할만한것이다.

11. 4. Intel/Sandia ASCI Option Red

OptionRed는 Intel Scalable System Division과 Sandia국립연구소에 의하여 공동으로 개발된 MPP체계이다. 이 체계는 1996년 12월에 Sandia국립연구소에서 전개되였다. 완전한 형태는 1987년 6월에 완성되였다.

11. 4. 1. Option Red의 구성방식

Option Red는 그림 11-3에 제시된 분산형기억기 MPP이다. 이 체계는 총 4608마디(매 마디들은 두개의 200MHz Pentium Pro처리기들을 가진다.)들을 리용하였는데 1.87flop/s의 최대속도와 51GB/s의 최대교차대역을 가진 594GB의 기억기를 리용하였다. 이 마디들가운데서 4536개는 콤퓨터마디들이며 32개는 봉사기마디들, 24개는 I/O마디들 그리고 2개가 체계마디이며 나머지는 호트에비이다.

이 체계는 1540개의 전원단과 616개의 호상련결지지판 그리고 640개의 디스크(용량이 1TB이상)를 가진다.

마디구성방식

콤퓨터마디들은 병렬계산응용프로그람들을 동작시키는데 리용된다. 봉사마디들은 log-in, 쏘프트웨어개발 그리고 기타 대화적인 조작을 지원하기 위하여 리용한다. I/O마디

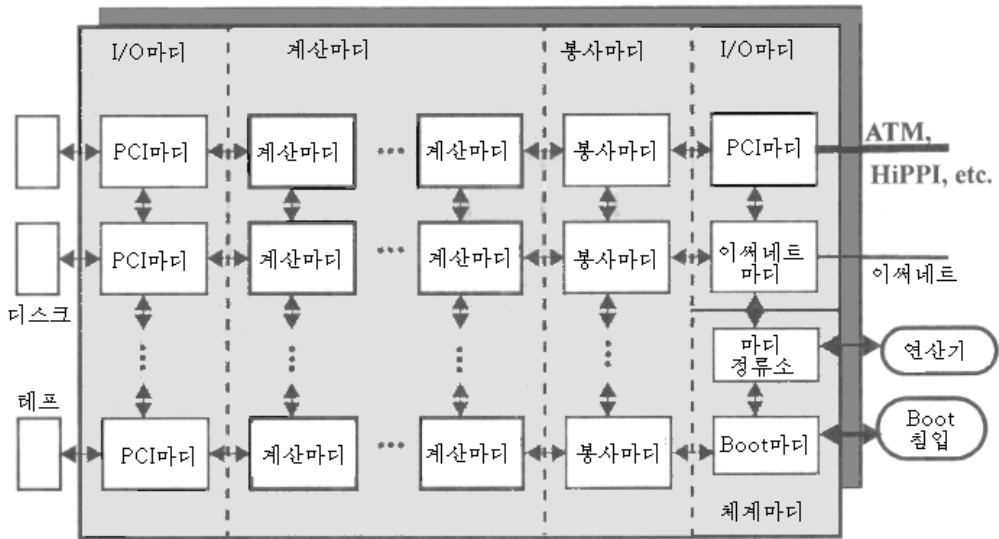



그림 11-3. ASCI Option Red체계블로크도식

들은 디스크, 테프, 망(Ethernet, FDDI, ATM, 등) 기타 I/O장치들이 접근하는데 리용된다.

다음과 같은 두가지 체계마디들이 있다. 즉

기동마디는 초기체계기동을 담당하며 단일체계화상지원을 위한 봉사기와 I/O마디들을 보장한다. 체계마디는 RAS성능을 지원하는데 리용된다.

콤퓨터마디들과 봉사마디들은 동시에 동작한다.

두 마디들은 그림 11-4 ㄱ)에 제시된것처럼 단일기판에서 동작한다. 두 SMP마디들은 NIC(Network Interface component)들을 련결하는 방법으로 사슬화된다. 한개의 NIC만이 련결지지판에 련결된다.

매 마디의 국부 I/O는 다음과 같은 부분을 포함한다.
직렬포구는 마디보존포구라고 불리우는 체계의 내부Ethernet에 련결되여 있으며 체계기동, 진단 그리고 RAS에 리용된다. 확장접속구는 마디검사를 보장한다.

기동지원하드웨어는 마디믿음성검사, BIOS, 마디고장진단과 조작체계적재에 필요한 기타 코드들을 포함하는 지우기ROM을 가지고 있다.

I/O와 체계마디를 위한 기판(그림 11-4 ㄴ), 쌍마디기판(그림 11-4 ㄱ)과 류사하다. 그러나 두개의 처리소자(1개마디), 단일국부모선과 한개의 NIC만이 있다.

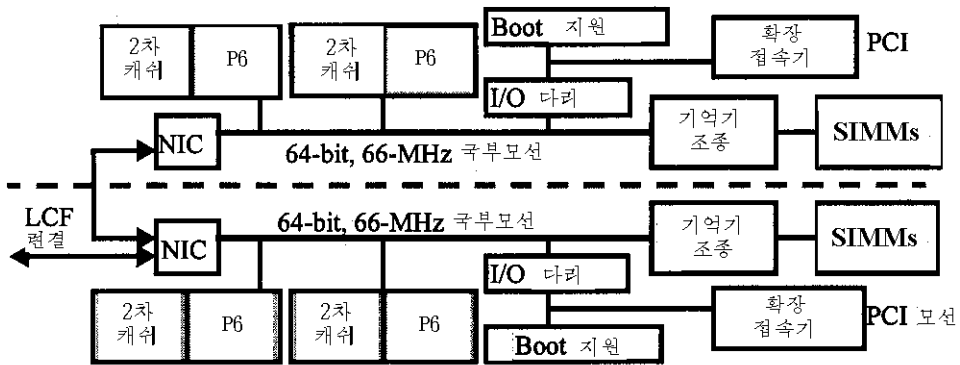
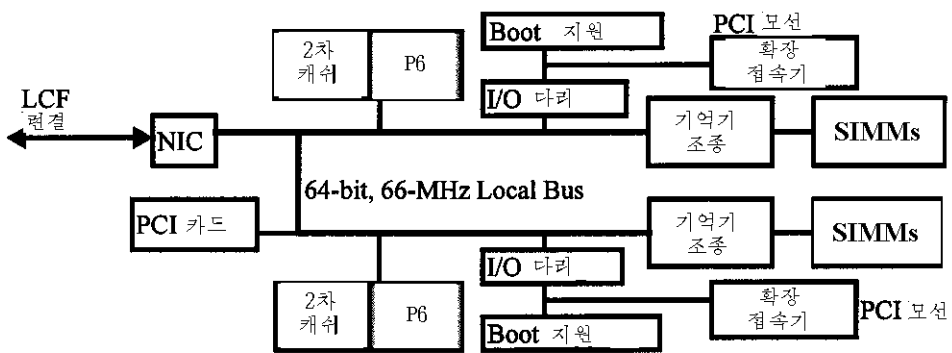
마디당 기억기용량은 32부터 286MB, 64MB, 1GB까지 증가한다. 133/s의 PCI카드수는 2부터 3까지 증가한다. 모든 I/O마디기판은 또한 RS23s, 이써네트(10Mb/s) 그리고 Fast_Write SCSI들과 같은 그러한 전면기판을 통하여 접근할수 있는 기판우의 기본 I/O설비를 가진다.

ㄱ) 계산과 봉사마디를 위한 쌍마디기판

ㄴ) I/O와 체계마디를 위한 단일마디기판

그림 11-4. 콤퓨터와 I/O 그리고 봉사기들을 위한 두개의 Option Red마디기판들

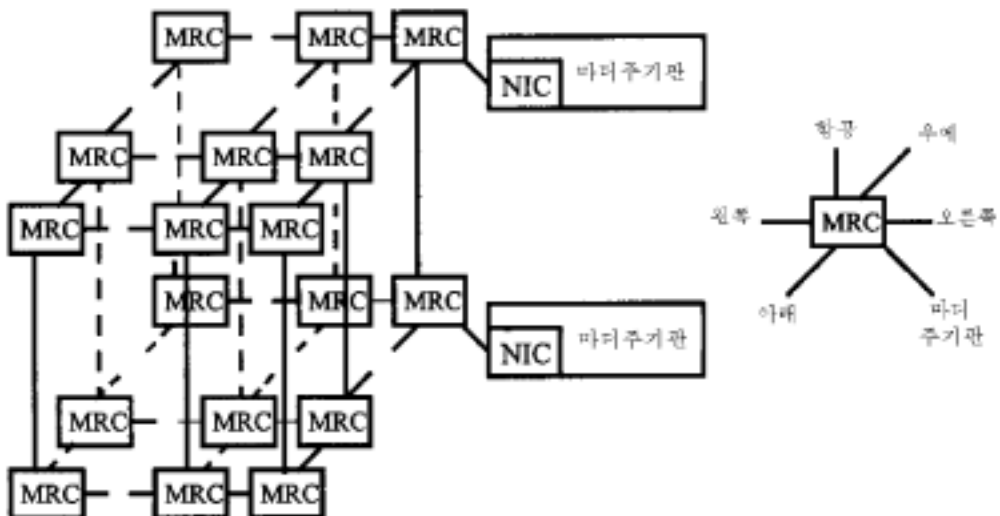



그림 11-5. Option Red 호상련결구성방식

체계호상련결

마디들은 호상련결된 설비(ICF)에 의하여 련결된다. ICF는 그림 11−5에 제시된것처럼 두 평면메쉬망위상구조를 리용한다.

매 마디기판은 마디기판에 있는 NIC를 거쳐 메쉬루팅요소(MRC)라고 불리우는 카스톰(ASIC)에 련결된다.

MRC는 6개의 쌍방향포구를 가지고 있다. 매 포구는 한 방향으로 최대 400MB/s, 그리고 완전 두방향으로는 800MB/s의 속도로 자료를 전송할수 있다.

4개의 포구는 그림 11−5의 좌, 우, 상, 하와 같이 내부면mesh련결에 리용된다. 이밖에 내부면련결을 위한 포구가 있다. 임의의 마디로부터의 통보들은 어느 한 면을 거쳐 벌레구멍형식을 따라 다른 마디로 간다. 이렇게 하면 지연이 감소되고 체계의 효률이 높아 진다.

11. 4. 2. Option Red의 체계쏘프트웨어

ASCI Option Red체계쏘프트웨어는 파라곤환경이 발전된 형이다.

체계, 봉사기, 그리고 I/O마디들은 파라곤조작체계를 실행한다. 이것은 OSF에 기초한 분산형Unix체계이다. 계산마디들은 Cougar라고 불리우는 LWR(light_weight_kernel)을 실행한다. 고속통신 Unix프로그람작성대면부, 병렬파일체계를 포함하여 이 두 체계를 대면시키기 위한 자원을 보장한다.

LWK(Light Weight Kernel)

LWK조작체계는 다음과 같은 설계특징을 가지는 PUMA(642)체계로부터 파생되였다.

- LWK설계는 수천개의 마디를 가진 MPP들을 효과적으로 지원하는 그런 기능이상으로 성능을 강화한다.
  이것은 일반적으로 OS봉사가 아니라 병렬계산에 요구되는 기능들만을 보장한다.
- TFLOPS체계에는 수천개의 계산마디들이 있다. Cougnr는 LWK가 리용하는 집합적기억기가 급격히 증가하는것을 막기 위하여 0.5MB이하의 기억기를 차지하도록 설계되여 있다.
- 설계는 통신망이 핵심부에 의하여 동작하여 조종된다고 가정한다. 보호검사와 통보립증을 위한 요구는 제기되지 않는다.
- LWK는 열린 구성방식을 가지며 사용자준위의 서고루틴들을 효과적으로 개발할수 있다.

그림 11−6에 제시한것처럼 LWK는 두개의 층으로 이루어 져 있다. 즉 프로쎄스조종스레드 (PCT)와 *Q*−핵심부로 이루어 져 있다.

매 마디는 여러개의 사용자과정, PCT와 *Q*−핵심부를 가지고 있다. *Q*−핵심부는 주소넘기기와 통신하드웨어에 직접 접근할수 있는 쏘프트웨어에 불과하다. 이것은 기본계산, 통신, 주소공간보호기능을 보장한다.

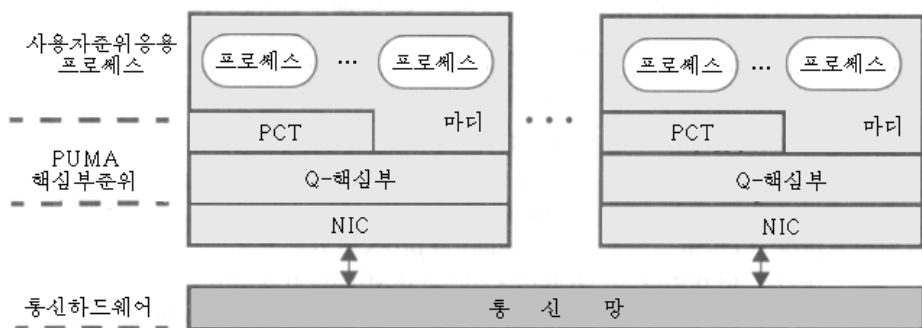



그림 11-6. LWK(Cowgar)환경의 층구조

PCT는 봉사라고 하는 그룹보호기능을 가진다. LWK환경은 신뢰도에 부분순서관계가 부여되였다는것을 전제로 한다.

즉 모든 요소들은 통신하드웨어가 정확하면서도 안전한 통신을 보장한다고 본다. 다시 말하여 하드웨어는 통보를 정확한 물리적마디로 확실하게 전송할것이다.

통보는 고장이 생긴 마디로 전송되든지 전송도중에 잘못되는 현상이 일어 나지 않을것이다. *Q*-핵심부는 하드웨어와 다른 마디의 *Q*-핵심부를 믿지만 PCT들이나 응용프로그람처리들은 믿지 않는다. PCT는 *Q*-핵심부와 다른 PCT들은 믿지만 프로세스들은 믿지 않는다. 응용프로그람처리들은 *Q*-핵심부와 PCT를 믿지만 다른 프로세스들은 믿지 않는다. LWK구성방식은 한개 준위의 자료구조가 다만 같거나 더 믿음성 있는 준위에 의하여 파괴될수 있다는것을 담보한다. 이것은 일련의 보호령역을 구성함으로써 달성된다. *Q*-핵심부령역은 한개 마디에 모든 물리적자원을 포함한다.

부분령역들은 PCT의 주소공간을 형성한다. 이 부분령역들은 또 여러개의 부분령역(매개가 프로세스에 대응하는)들을 포함한다. 매개 프로세스는 자기 령역에만 직접 접근될수 있다.

Q-핵심부를 파괴시킬수 없는 프로세스는 PCT를 파괴시킬수 없다. 핵심부를 두 층으로 나누면 여러가지 우점이 있다.

이렇게 하면 첫째로, 이식성을 증가시킨다.
다른 MPP인 경우에는 *Q*-핵심부코드의 대부분을 다시 작성하여야 하지만 PCT코드의 대부분은 그대로 이식할수 있다.

둘째로, 이와 같이 층들로 나누면 기능관계, 업무들을 분리시킬수 있다. 더 중요하게는 *Q*-핵심부가 물리적자원접근조종을 담당하며 PCT는 이러한 접근을 관리하는 방법으로 배정된다.

PUMA의 설계가들은 조종과 관리를 특별히 정의하지 않는다.

대략적으로 말하면 관리라는것은 과제수행을 위하여 무엇을 어떻게 하겠는가를 지시하는것이며 조종이라는것은 무엇을 어떻게라는 결심을 실제적으로 실행하는것을 의미한다. 실례로 시간구간의 크기를 선택하는것, 어느 과정이 통신의 목적대상으로 되는가를 결정하는것, 프로세스분담방법을 선택하는것 등이 관리조작의 실례들이다. 시간구간을 실제적으로 분리(실행)하는것, 통보의 목적대상이 유용한가를 검열하는것, 분할방법을 실현하는것 등은 조종조작의 실례들이다.

PUMA설계가들은 조종조작이 관리조작보다 훨씬 더 자주 일어 나며 관리방법은 조종기구보다 더 자주 변한다고 본다. LWK를 두개의 준위로 나누면 관리방법에 큰 영향을 주지 않고서도 조종조작을 쉽게 최량화할수 있으며 같은 *Q*−핵심부에서 각이한 PCT가 다중과제 혹은 단일과제를 실행할수 있도록 한다.

통보문넘기기

ASCI Option Red체계는 MPE, NX 그리고 통보문넘기기를 위한 입력을 지원한다. MPI는 체계에서 표준서고이다. NX는 파라곤과 거꿀방향일치성을 보장하기 위하여 실행된다. 파라곤상에서는 많은 응용프로그람들이 NX통보문넘기기서고를 리용한다.

입력은 가장 효과적이며 낮은 준위 통보문넘기기서고를 보장한다. 입력이라는 개념은 PUMA조작체계에서 처음으로 제기된것이다. 그것은 통보문넘기기에서 기억기복사부가처리를 제거하는데 리용할수 있다. 입력을 사용하는 통보문넘기기는 7.4.3에서 고찰한 바와 같은 그런 사용자준위통신메하니즘이 아니다. 핵심부교차는 여전히 필요하다.

입력은 통보문전송을 위하여 다른 과정들에 개방하는 목표프로쎄스의 주소공간의 일부이다.

통보문을 송신하기 위하여 송신측 프로쎄스는 다음과 같은 표본을 가진 핵심부루틴을 실행한다.

```
Send_user_msg(
Void*buf,   //송신통보문완충기의 시동
Size_t len,   //송신통보문크기
Int tag,   //통보문꼬리표
Proc_id dest,   //목적지프로쎄스 ID
Portal_id portal,   //목적지portal의 첨수
Int *flag   //msg가 송신될 때 증가하여야 하는 기발
)
```

이것은 핵심부가 통보문넘기기를 위하여 필요한 정보를 기록하자마자 귀환하는 차단접근이다. 원천핵심부는 송신측프로쎄스 ID, 목표측프로쎄스 ID, 통보문길이와 꼬리표, 목표입력을 포함하는 통보문머리부를 목표핵심부에 보낸다.

그러나 통보문완충기주소와 기발은 보내지 않는다. 머리부가 도착하면 목표핵심부는 그것을 번역하여 통보문이 입력에 기억될수 있는가를 검사한다. 그다음에 그것은 통보문본체가 목표입력에 송신되도록 한다.

모든 통보문들이 목표입력에 기억된후에 목표핵심부는 입력서술자에 있는 한 비트를 설정하여 새 통보문의 도착을 알린다.

통보문이 전송되면 원천핵심부는 기발값을 하나 증가시킨다. 송신측 프로쎄스는 그 기발을 송신통보문을 언제 다시 리용하는것이 안전한가를 결심하는데 리용할수 있다. 명백한 루틴은 통보문을 접수하기 위하여 접근될 필요가 없다.

수신측 프로쎄스는 통보문이 도착하였다는것을 알아 내는데 입력서술자를 리용할수 있다. 프로쎄스는 또한 그 입력들가운데서 임의의것에 통보문이 도착하면 신호가 발생하도록 할수 있다.

１１．５．NAS 병렬성능평가기준의 실험결과

유효한 병렬콤퓨터를 설계할 때 체계설계자는 많은 문제점들에 부닥친다.
기본문제점은 다음과 같다.
주요응용프로그람들을 지원하는 구성방식적특성은 무엇인가?
다음파라메터들은 중요한 응용특성들의 부분적인 목록이다.

- 주기억요구(MB)
- I/O속도(MB/s로)와 I/O자료량(MB)
- 통신지연시간(t_0 μs)과 대역너비(r_∞MB/s)
- 평균병렬성, 순차적인 병목현상(Amauhl 법칙), 립도 그리고 부하불균형 등

이 파라메터값들은 다른것들가운데서 응용프로그람, 문제크기, 기계크기, 처리기속도에 의존된다.
이 절에서는 과학적모의, 계산류체력학, 신호처리응용에 대한 세개의 NAS성능평가실험들의 구성방식적인 련관관계를 론의한다.
이 연구결과들은 자주 제기되는 세가지 질문들에 부분적인 해답을 준다.

- 기억기가 얼마나 큰가?
- 통신부분체계가 얼마나 빨라야 하는가?
- I/O가 얼마나 빨라야 하는가?

１１．５．１．NAS 병렬성능평가기준의 실험결과

NAS병렬성능평가실험(혹은 NPB)들은 이미 3.1.2에서 론의하였다.
표 11-7에 Gflop의 총 개수로서 8개의 NPB성능평가실험들의 문제크기가 제시되였다. [593]에서 Sun 등은 NPB MG와 FT병렬성능평가실험들을 특징 짓고 병렬콤퓨터구성방식을 설계하는데서 련관관계를 론의하였다. 그 특징은 1.3.3에서 위상병렬모형에 기초하고 있다.
우리는 매 성능평가실험을 다단계렬로 나누고 매 다단계에 대하여 기억기, I/O, 통신요구사항들을 규정하였다. 그러면 각이한 기계크기 *n*과 문제크기 N을 가진 매 성능평가실험에 대하여 다음과 같은 값들이 얻어 진다.

- 평균병렬성, 순차적인 몫의 비률(Amdahl법칙), 립도 *w*, 부하불균형성 δ 등을 포함하는 작업부하구조
- 마디당 성능평가실험에 요구되는 기억기(MB)
- I/O속도(MB/s로)와 I/O자료량(MB)
- 통신페턴, 통보길이, 지연시간(t_0, μs)과 대역(r_∞, MB/s)요구사항들을 포함하는 통신요구사항

표 11-7	NAS 병렬성능평가실험문제크기		
성능평가실험문제크기	총 작업부하(Gflop)		
	Class A	Class B	Class C
Embarrassingly parallel (EP)	26.68	100.9	N/A
Multigrid(MG)	3.905	18.81	155.5
Conjuagte Gradient(CG)	1.508	54.89	
3D FFT PDE(FT)	5.631	71.37	N/A
Integer Sort(IS)	0.7812	3.150	N/A
LU Solver(LU)	64.57	319.6	2039
Pentadiagonal Solver(SP)	102.0	447.1	
Block Tridiagonal Solver(BT)	181.3	721.5	

우리는 클라스 C문제크기에 대한 MG성능평가실험들과 FT에 대한 결과들을 설명한다.이 결과들가운데서 어떤것은 처리기속도를 파라메터로 리용한다.

우리는 항상 일정한 속도(Mflop/s)를 념두에 둔다. 현재와 다음세대극소형처리기들은 200부터 1000Mflop/s의 최대속도대역을 가진다. 그러나 흔히 속도는 최대속도의 5~10%에 불과하다. 따라서 우리는 다음과 같은 세가지 경우에 대한 결과들을 설명한다. 다시 말하여 처리기속도는 10, 50 혹은 100Mflop/s이다. 또한 이 방법들은 다른 값들에 대하여서도 유용하다.

11. 5. 2. 초걸음구조와 립도

FT와 MG 성능평가실험의 다단계구조는 그림 11-7에 제시되였다.

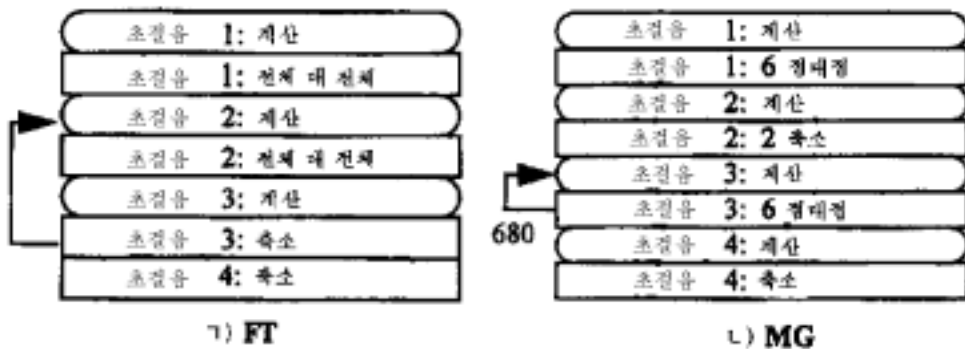



그림 11.7. FT와 MG의 간단한 구조들

매 타원형도식은 계산단계, 4각형도식은 통신단계를 의미한다. 화살표는 반복을 의미한다.

실레로 MG성능평가실험은 4개의 다단계로 구성되는바 세번째 단계는 680번 반복된다. 첫째 단계의 통신단계는 6개의 점대점으로 구성되며 둘째 단계의 통신단계는 2개의

작은 4개의 점대점으로 구성된다.

1.3.3에서의 단계병렬모형에 따르는 다단계의 거침성은 한개의 다단계에서 한개의 마디에 의하여 수행되는 작업부하(즉 Gflop)이다.

FT와 MG에 대하여 우세한 다단계의 거침성 w는 표 11-8에 제시되였다.

표 11-8 FT와 MG에서 우세한 다단계의 거침성(Gflop)

마디	2	8	32	128	512
FT	10036	1221	305	76	19
MG	285.2	71.3	17.8	4.5	1.1

거침성은 서로 다른 성능평가실험들사이에서 심하게 변하는바 기계크기가 증가함에 따라 감소한다.

다단계모형은 다음과 같은 두가지 실제적인 점들을 보여 준다.

- 단계병렬모형은(1.1.3) 모든 NAS성능평가실험들을 연구하는 효과적인 방법이다. 다단계구조는 많은 세부들을 추상화하여 버림으로써 매우 단순화된다.
- 매 성능평가실험프로그람은 계산작업량의 대부분이 실행되는 하나 혹은 두 세개의 우세한 다단계를 가진다. 첫 둥그리기근사화로서 그 주요다단계들에 대해 검증할수 있다. 실례로 세번째 단계는 그것이 클라스 C의 문제크기에 대하여 680번 반복하므로 MG에서의 우세한 단계이다.

11．5．3．기억기，I/O 및 통신

임의의 병렬콤퓨터체계설계에서 결정하여야 할 관건적인 파라메터들은 마디당 물리적기억기용량이다. 이 파라메터들은 또한 주소공간에 대한 어떤 일정한 최소크기를 요구하는 방법으로 처리기설계에 영향을 줄수 있다.

기억기요구들

기억기요구는 기계크기의 함수로써, 마디당 요구되는 기억기용량으로써 정의할수 있다. 이것은 다음과 같은 모든 형태의 기억기사용을 포괄한다. 즉 자료기억 <입력>/<출력>, 림시기억, 통신통보문완충기들, 등등.

FT와 MG에 대한 마디당 기억기요구사항은 그림 11-8에 제시되였는바 거기에는 세가지 련관관계가 서술되였다.

- 첫째로, 클라스 C성능평가실험는 큰 문제크기를 가지며 현재의 MPP들에 있는 단일마디에 의하여 리용할수 있는 기억기보다 더 큰 기억기를 요구한다. 사실상 FT와 MG성능평가실험들은 일부 처리소자들의 물리적주소공간크기보다 더 큰 3.5～4.3GB의 주기억을 요구한다.
- 둘째로, 마디당 요구되는 기억기는 더 많은 마디들이 리용될 때 거의 선형적으

로 감소하며 통보문넘기기에 리용되는 기억기는 전체 기억기가운데서 적은 몫을 차지한다. 실례로 MG를 실행시킬 때 8개의 마디들이 리용된다면 마디당 요구되는 총 기억기와 마디당 요구되는 통보문완충기는 각각 457MB와 4.26M B이다. 그러나 64개의 마디의 경우에는 각각 58.2MB와 1.08MB이다.

- 셋째로, 그림 11-8은 NAS클라스 C성능평가실험를 실행시키는 MPP에 대하여 얼마나 많은 마디들이 요구되는가를 보여 준다. 실례로 대부분의 MPP들은 응용프로그람에 의하여 리용될수 있는 1.28MB보다 적은 마디당-기억기를 가진다. FT와 MG성능평가실험들을 실행시키자면 적어도 64개의 마디를 리용하여야 한다.

I/O 요구들

I/O량은 매 성능평가실험에 대하여 맨 처음에 계산한다.

다음으로 요구되는 I/O속도를 유도한다. I/O속도는 I/O량, 처리기의 개수, 처리기속도에 의존한다. 우리는 다음과 같은 균형적인 가정을 통하여 I/O속도를 유도한다. 즉 총체적인 I/O시간은 통보문넘기기부가처리를 무시한 총계산시간과 같아야 할것이다.

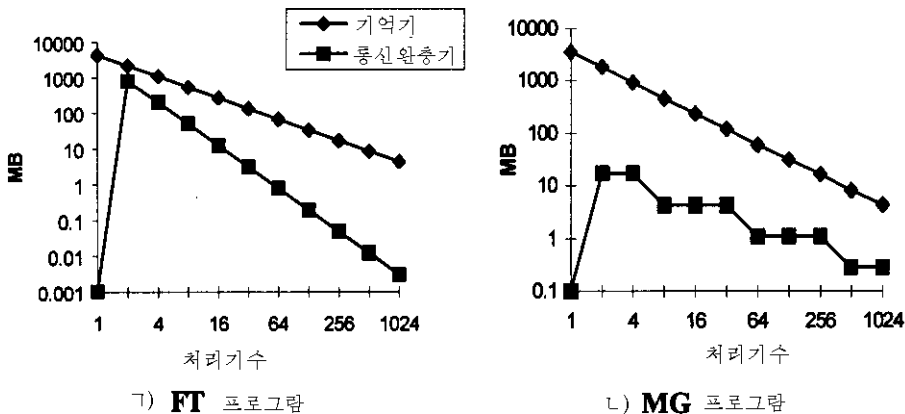



그림 11-8. 2개의 NAS프로그람에 대한 마디기억기요구들

표 11-9는 2~2048개의 마디들을 거쳐 FT와 MG에 대하여 요구되는 I/O속도가 제시되였다.

매 성능평가실험에 대하여 세개의 행은 세가지 경우를 표현하는바 이 경우에 처리기당 속도는 각각 10Gflop/s, 50Mflop/s, 100Mfop/s이다. 요구되는 I/O속도는 더 많은 마디들을 리용할 때든가 처리기들이 더 빨라 질 때 뚜렷하게 증가한다.

실례로 처리기속도가 10Mflop/s이면 요구되는 I/O속도는 기계크기가 2개의 마디 또는 32개의 마디일 때 MG에 대하여 각각 0.14MB/s, 2.29MB/s이다.

그러나 처리기속도가 100Mflop/s까지 증가할 때 요구되는 I/O속도들은 1.4MB/s 또는 22.9MB/s까지 증가한다.

표 11-9 FT 프로그람과 MG 프로그람에서 요구되는 I/O

NAS코드	처리기 속도	마 디 수					
		2	8	32	128	512	2048
FT	10Mflop/s	0.05	0.21	0.85	3.4	14	56
	50 Mflop/s	0.27	1.06	4.25	17	68	272
	100 Mflop/s	0.53	2.12	8.5	34	136	544
MG	10 Mflop/s	0.14	0.57	2.29	9.39	38.7	164
	50 Mflop/s	0.7	2.82	11.47	46.94	193.6	822
	100 Mflop/s	1.4	5.65	22.9	93.9	387	1643

현재의 디스크와 RAID기술로써 직렬 I/O(단일디스크 혹은 단일RAID)는 소형으로부터 중간크기의 기계(실례로 64개 마디이하)나 느린 처리기들에는 아마 충분하리라고 본다.

그러나 기계크기가 64개 마디보다 커지고 처리기는 더 빨라 지며 균형적인 체계를 유지하는데 병렬I/O가 요구된다.

그러나 모든 NAS성능평가실험들은 정규화된 I/O접근패턴들을 가지며 파일이 다중디스크를 거쳐 단번에 넘어 가는 병렬I/O를 실현하기 어렵지 않다는것을 보여 준다.

통신요구

다른 중요한 인자는 통신부분체계가 얼마나 빨라야 하는가 하는것이다. 두가지 통속적인 파라메터들은 통신지연시간(혹은 시동시간) t_0과 두 마디사이에서 점대점통보문을 보내기 위한 접근대역 r_∞이다.

우리는 이 파라메터들을 다음과 같이 결정한다. 다시 말하여 우선 통신시간을 다단계구조와 처리속도로부터 계산할수 있는 다단계당 계산시간과 같도록 설정한다.

다음에 방정식 (1.11)을 리용하여 통신시간으로부터 대역과 지연요구를 도출한다.

대역요구들은 그림 11-9에 제시하였다. 지연시간효과가 작은 경우가 발생할수 있다.

요구되는 대역은 처리기속도에 비례하여 증가하는것이 분명하다.

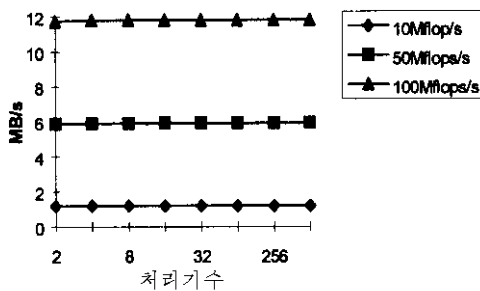
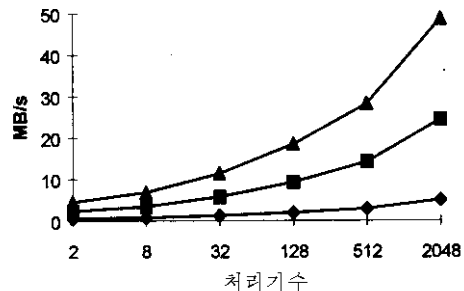



그림 11-9. NAS 프로그람들에서 요구되는 통신대역들

그러나 요구되는 대역은 FT곡선이 평탄한 경향성을 보여 주듯이 더 많은 처리기들이 리용될 때 필연적으로 증가하는것은 아니다.

현재 및 다음세대의 처리기들에 대하여 2000개의 처리기들까지는 40−50MB/s의 대역이면 충분하다.

구성방식적련관관계

종합해 보면 NAS FT와 MG성능평가실험들은 다음과 같은 구성방식적련관관계를 가진다는것을 알수 있다.

즉 수 Mflop로부터 수천 Mflop에 달한다.

C NAS성능평가실험들을 실행시키려면 처리기당 기억기가 적어도 마디당 64~128MB가 되여야 한다.

I/O속도요구는 처리기속도와 기계크기가 증가함에 따라 수백MB/s의 범위에서 증가한다.

NAS결과들은 병렬 I/O를 시사해 준다. 더우기 지연시간은 Berkeley결론과는 반대로 대역너비보다 덜 중요하다.

마디당 50MB/s의 대역너비이면 매 처리기는 100Mflop/s의 속도를 유지하는 2000개까지의 처리기로서 충분하다.

우리는 독자들에게 이 결과는 통보문넘기기기계에서 NPB를 리용한 한가지 연구에 기초한것이라는것에 대하여 주의를 준다.

각이한 프로그람다중모형을 가지는 각이한 가동환경에서 각이한 성능평가실험들에 대하여서는 결과와 결론이 심히 차이날수 있다.

11.6. MPI 및 STAP성능평가기준의 실험결과

마지막절에서 우리는 남부캘리포니아종합대학(USC)의 Hwang의 연구집단과 홍콩종합대학(HKU)에서 1994년부터 1997년사이에 달성한 MPI와 STAP성능평가실험결과들가운데서 일부를 소개한다.

11.6.1. MPI성능측정

통보문넘기기형대면부(MPI)에서 NUMA, 다중콤퓨터 혹은 워크스테이션의 클라스들은 프로그람작성할 때 통보문넘기기함수를 규정하기 위한 일반적인 통용통신표준으로 되였다. MPI기능의 세부들은 14장에서 취급하게 될것이다.

MPI 성능평가실험결과들은 세가지 MPP 즉 [332]에서 처음으로 보게 된 T3D, SP2, 파라곤들에서의 성능평가실험에 기초한다.

집체적인 통보화조작은 프로쎄스들로 이루어 진 그룹을 포함하며 인수들을 정합하여 같은 집체적인 통신루틴을 접근시키는 같거나 다른 마디들이 갖추어 져 있다.

전형적인 집체적조작들은 방송, 모으기, 산란, 전체 교환, 차단, 감소, 주사(앞붙이)등이다. 이 조작들은 사용자가 자기의 응용프로그람코드를 설계하는데 공통인 대면부를 보

장한다.

MPI실행들

MPI의 여러가지 실행은 CHIMP/MPI [19], LAM[462], mpi++[357], MPICH[423]과 같은 공개적인 령역에서 가능하다. 많은 MPP판매자들은 그들자신의 기계구성방식에 관하여 최량인 고유한 실행방식들을 창조하였다. 실례로 CRI/EPCC CPI[122]는 T3D우에서 리용할수 있다. MPICH는 SP2과 파라곤우에서 적합한것이다. MPI실행방식들은 표 11-10에 종합적으로 보여 준 집체적조작의 풍부한 모임을 제공하여 준다.

우리의 실험에 의하면 리용되는 자료구조는 페지오유를 피할수 있도록 매 마디기억기에 충분히 맞을 정도로 작게 만들어 졌다.

검사프로그람은 표준C와 MPI원시형으로 작성되였다. 여기에 기계특유의 서고함수든지 임의의 번역코드가 리용된다. 특히 매 기계에 쓸수 있는 가장 좋은 콤파일러선택권이 항상 리용된다.

검사조건들

MPI_Wtime()함수를 리용하여 벽시계시간을 측정한다. 검사프로그람은 예비동작효과를 배제하도록 세번째 반복할 때 시작되는 박자에 맞추어 22번이상 반복실행한다. 모든 처리로부터 최소시간, 최대시간, 평균시간을 모은다. 그 결과들을 분석하기 위하여 우리는 최대시간에 주의를 집중한다. 왜냐하면 우리는 이것이 기계에서의 모든 처리들은 조작이 끝난 상태를 반영한다고 보기때문이다.

통보의 총 길이

$f(m, n)$을 n개의 마디를 포함하는 집체적조작에서 통보 총길이라고 하자.

이것은 집중통신조작에서 마디들의 모든 쌍사이에 전송되는 모든 통보들의 합과 같다. 실례로 방송조작에서 m은 원천마디로부터 방송되는 통보문의 길이(B로)이다. 따라서 $n-1$개의 목표들이 같은 통보문을 수신할것을 요구하므로 $f(m, n)=m(n-1)$이다. 류사하게 산란, 모으기, 감소, 주사조작들에 대하여서도 $f(m, n)=m(n-1)$이다.

전체 교환조작에 대하여서는 $f(m,n)=mp(n-1)$이다.

표 11-10 MPI집체적조작들

연산	기능서술
MPI_Bcast	통보문을 같은 그룹에 있는 모든 처리들에 방송한다.
MPI_Barrier	모든 프로쎄스들이 이 루틴에 도달할 때까지 차단
MPI_Alltoall	자료를 전체에서 전체 프로쎄스에로 보낸다.
MPI_Gather	프로쎄스그룹에서 값들을 수집한다.
MPI_Reduce	모든 프로쎄스들에서의 값을 단일값으로 축소한다.
MPI_Scan	프로쎄스수집의 부분축소를 계산
MPI_Scatter	그룹에서 자료를 한 과제에서 모든 과제에로 보낸다.

집중통신조작을 평가하기 위하여 4가지 성능측정기준을 표 11-11로 통합하였다.

표 11-11　　　　집중통신측정기준

전체 통보시간(μs)	$T(m, p) = t_o(n) + d(m, n)$
시동지연시간(μs)	$t_0(n)$
이행지연(μs)	$d(m, n)$
집합대역너비(Mbyte/s)	$R_\infty(n)$

우리는 통신머리부로써 총 통보화시간방정식 (3.4)에서 $T(m, n)$을 정의하였다. 이 시간은 두개 즉 시동시간 $t_0(n)$과 전송지연 $d(m, n)$으로 나눌수 있다.

이 표본화에 기초하면 모든 시간과 파라메터들은 마이크로초로 측정된다. 점근대역 $r_\infty(n)$은 방정식 (3.3)에서 정의하였다.

우리는 다음과 같은 일반공식에 의하여 $r_\infty(n)$와 관련되는 집합대역너비 $R_\infty(n)$을 정의하였다. 즉

$$R_\infty(n) = r_\infty(n) \times f(m, p) \qquad (11.1)$$

점대점통신인 경우에 방정식 (11.1)은

$$R_\infty(n) = r_\infty(n) \qquad (11.2)$$

으로 쓸수 있다.

전체 교환조작에 대하여서는

$$R_\infty(n) = n(n-1)\ r_\infty(n) \qquad (11.3)$$

기타 집체적조작들(방송, 모으기, 산란, 주사 등)에 대하여서는

$$R_\infty(n) = (n-1)\ r_\infty(n) \qquad (11.4)$$

이다.

11. 6. 2. MPI지연시간과 집합대역너비

방송과 전체 교환조작에 대한 세개의 기계에 대하여 시동지연시간 $t_0(n)$은 그림 11-10에 제시하였다.

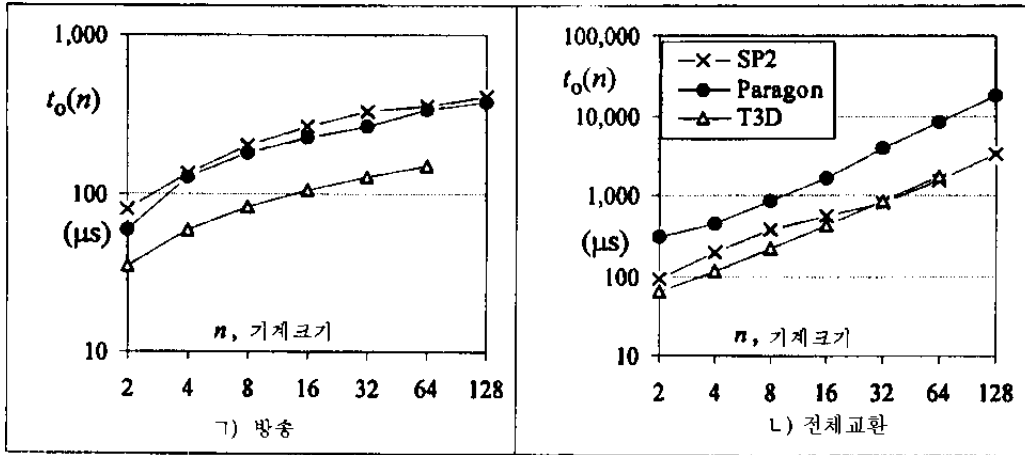



그림 11-10. 2~128개의 마디를 가지는 기계를 거치는 두가지 MPI집체적조작의 시동지연시간들

이 지연시간은 기계크기 n에 따라 단조증가한다. 시동지연시간은 모으기, 산란, 전체 교환조작의 경우에도 기계크기에 따라 선형적으로 증가한다. 지연시간은 큰 기계가 리용될 때 방송, 주사, 감소, 차단조작들에 대하여서는 로그함수적으로 증가한다.

T3D에서 지연시간들

T3D는 주사조작을 제외한 모든 집체적조작들가운데서 가장 작은 시동지연시간을 가진다.

T3D는 고속통보넘기기를 위한 특수하드웨어를 갖추고 있으며 작은 망지연시간(SP2에 대하여 125ns, 파라곤에 대하여 40ns인것과 정반대로 hop당 20ns) 그리고 원격기억기접근지연시간을 가리우기 위한 미리꺼내기대기렬과 원격처리기보관을 리용하므로 작은 지연시간을 가진다.

파라곤과 SP2의 지연시간들

방송조작에 대하여 SP2은 파라곤보다 약간 더 큰 지연시간을 가진다(그림 11-10ㄱ)).

파라곤은 전체 교환, 산란, 모으기, 감소조작들에서 가장 긴 지연시간을 가진다. 그러나 주사조작에 대하여서는 T3D보다 훨씬 더 짧은 지연시간을 가진다. 큰 기계를 거쳐 집체적조작들이 통보넘기기할 때에 파라곤은 다른것들보다 훨씬 더 큰 지연시간을 가진다. 작은 기계를 거치는 집체적조작들에 대하여서는 SP2이 전체 교환, 산란, 모으기조작들의 경우에 지연시간에서 두번째 자리를 차지한다.

시간측정결과의 분석

통보당 n=32마디, m=1KB를 가진 세개의 MPP들에 대한 6개의 집체적기능들의 실제적인 성능은 그림 11-11에 제시되였다.

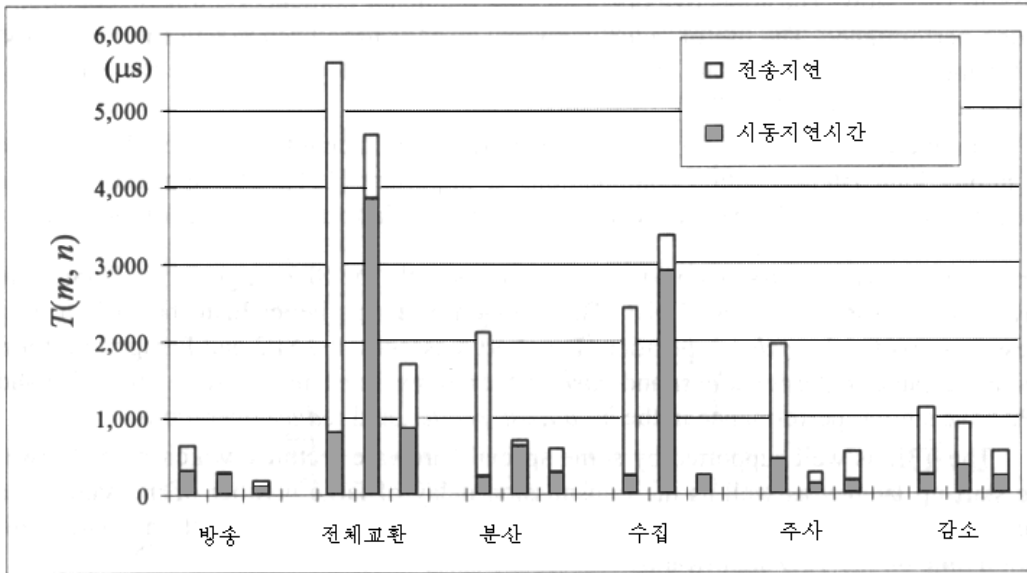



그림 11-11. MPI집체적조작에 대한 총통보시간의 약화

매 막대기의 검은 부분은 시동지연시간을 의미하며 흰 부분은 전송지연시간을 의미한다. 같은 집체적조작들에 대하여 세가지 기계의 성능순위는 막대기높이에 반비례한다. 분명히 전체 교환은 기능을 완성하는데 가장 긴 시간이 요구된다. T3D는 방송, 모으기, 감소조작들에서 가장 작은 시동지연시간을 가진다. 파라곤에서의 지연시간은 수행되는 각이한 집체적기능에 따라 극적으로 변한다. 실례로 전체 교환과 모으기조작들에서 파라곤은 SP2과 T3D계수기부분보다 약 4~15배 더 큰 3587μs와 2918μs의 지연시간을 가진다.

주사의 경우에 파라곤은 지연시간이 더우기 작다는것을 보여 준다.

통보길이가 증가함에 따라 전송지연시간(막대기 흰 부분)이 선형적으로 증가한다.

총 대역너비

총 대역은 각이한 크기를 가진 기계들이 실제적인 성능에 대한 한계를 설정할수 있도록 한다. 64개 마디경우에 T3D, 파라곤, SP2에 대한 전체 교환의 총 대역은 각각 1.745, 0.879 그리고 0.818GB/s이다.

집합대역너비는 집체적통보전송조작에서 자료전송속도를 정량화하는 더 좋은 성능평가실험을 규정한다. Hockney[314]에 의한 접근대역은 점대점통신에만 리용할수 있다.

여러가지 MPI집체적함수들(방정식 11.1부터 11.4까지)에 대하여 정의된 총 대역은 이런 목적에서 리용할수 있다.

능동적인 통보를 가진 갱신된 MPI

세개의 MPP들에서 MPI실행은 여전히 매우 느리다. 집체적조작들은 MPP통신성능에서 주요병목현상들이다. MPI에서의 응용통보의 리용은 MPI-AM[177]과 MPI-FM[424]에서 실현되였다. 자료미리꺼내기는 또한 통신지연시간을 줄이는데 리용할수 있다[124, 150, 522].

MPP는 구체적으로 병렬성되여 있어야 하며 응용프로그람코드들은 하드웨어가동환경이나 망의 위상적구조에 무관계하게 실행되여야 한다.

구성방식적속성들

통보길이를 고려하면 SP2은 1KB보다 더 짧은 통보에 대해서는 파라곤을 릉가한다.

파라곤은 감소조작을 제외하면 긴 통보문의 경우에 SP2보다 기능을 더 잘 수행한다. 통보문이 짧은 경우에 SP2과 파라곤이 방송과 차단조작들에서는 거의 같은 기능을 수행한다. 통보문이 긴 경우에 대해서도 T3D와 SP2들은 방송, 산란, 감소조작들에서 류사한 성능을 가진다.

T3D는 집중통신에서 시동지연시간뿐아니라 다량의 통보전송지연시간을 줄이는 일부 특수한 하드웨어특징에 의하여 훌륭히 지원된다.

장치배선에 의한 차단은 T3D에서의 동기시간을 뚜렷히 감소시킨다. 파라곤은 집체적통보문넘기기에서 NX부가처리로부터 긴 지연시간에 대하여 짧은 통보를 다루는데 미약하다.

지연시간에서의 이러한 파동은 특히 파라곤에서 전체 교환과 모으기조작을 수행할 때 실제적으로 나타난다.

SP2은 망대역이 제한되여 있는 경우에 긴 통보들을 다루는데 미약하다.

이상의 MPI평가결과들은 통보문넘기기다중콤퓨터에서 병렬응용프로그람을 개발하려는 사람들에게 유용할것이다. 집체적지연시간과 통보전송시간표형식과 서술한 수값자료들은 집체적통보문넘기기를 리용하는 SPMD, MPMD계산의 절충연구에 리용할수 있다. 지연시간과 통보넘기기지연시간은 [656]에서 보는바와 같은 MPP성능을 예측하는데 리용할수 있다.

예측과정의 자세한 내용들을 과제로 남겨 둔다.

１１．６．３．MPP 들의 STAP 성능평가기준

현존 MPP들의 성능은 [334, 634, 656]에서 고찰한 STAP성능평가실험에 의하여 평가되였다.

아래에 제시된 매체 덩어리는 서술된 결과들에 기초하고 있다.

STAP성능평가실험을 통하여 배운 내용들은 다른 리용자들이 자기의 병렬프로그람을 MPP에서 개발할 때 참고하도록 한다.

STAP프로그람의 병렬성

우리는 단계-병렬형을 리용하여 STAP성능평가실험을 병렬성하였다.

이 방법은 서로 다른 MPP가동환경에 병렬STAP프로그람들의 이식을 가능하게 하였다. 우리는 MPT를 모든 내부처리기통신함수를 실행하도록 선택하였다. 위상병렬모형을

따르면 매 처리기는 각이한 자료부분모임에 대하여 값 SPD프로그람을 실행한다.

마디프로그람

그림 11-12에 세가지 MPP들가운데 임의의것에서의 전형적인 병렬STAP성능평가실험프로그람의 자료흐름과 조종흐름을 제시하였다. 위상병렬모형을 따를 때 여러가지 자료부분모임우에서 매 처리기가 같은 SPMD프로그람을 실행한다.

매 마디프로그람은 본질상 세개의 기본단계로 구성된다. 즉 도플러처리(DP), 성능평가실험형성(BF), 목표검출(TB)들이다.

입력자료립방체는 RNG차원을 따라 P개의 꼭 같은 쪼각(회색으로 표시한 수직쪼각)으로 분리된다. 매 마디에는 그림 11-12에 제시된것처럼 자료립방체의 한쪼각만을 처리할것을 요구한다. 그러면 부분으로 분할된 매 쪼각들은 모든 마디들에 의하여 동시에 처리된다.

한 단계안에서 모든 부분프로그람들이 실행될 때 처리기들은 절환망을 통하여 통보문넘기기에 의해 호상작용하거나 자료를 교환하도록 해준다.

다단계Omega망은 SP2에서, 20메쉬망은 파라곤에서, 3D망은 T3D에서 리용된다.

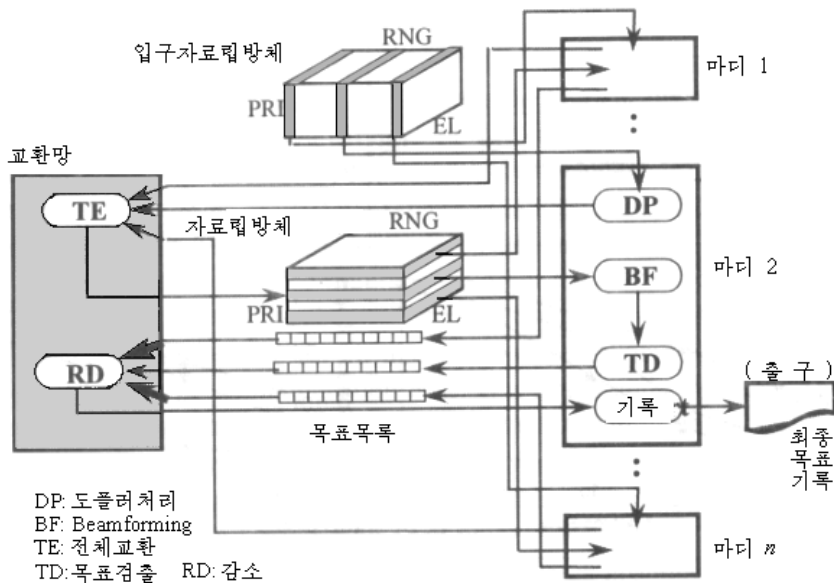



그림 11-12. 매 마디프로그람에서 자료조종형식과 망을 통한 마디사이 집중통신

일정한 속도

일정한 속도는 그림 11-13에 제시된것처럼 기계가동환경과 배렬된 마디수에 따라 변한다. 세가지 기계들가운데서 128개 마디의 임의의 STAP프로그람들을 실행할 때 그 어느 기계도 10Tflop/s를 초과하지 않는다는것을 관측하였다.

일반적으로 SP2이 모든 크기의 기계들가운데서 최대인 일정한 속도에 도달하였으면 그다음은 APT프로그람의 속도결과를 보여 준다. APT프로그람은 한개의 SP2마디에서 60Mflop/s, 128개 SP2마디에서는 4.5Gflop/s를 보장하였다.

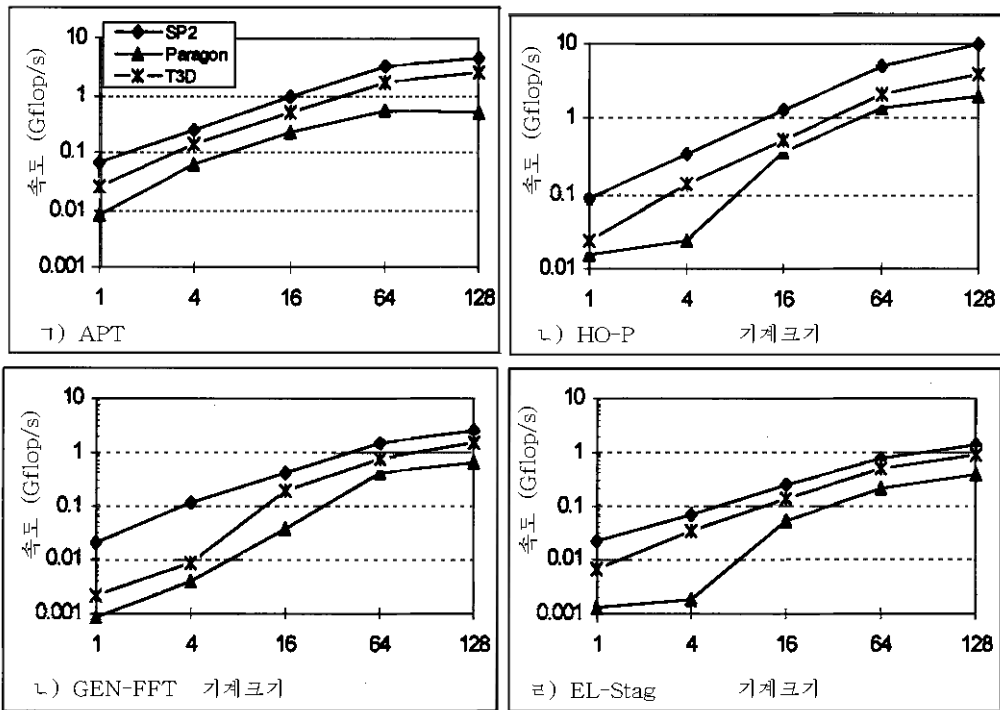



그림 11-13. 네가지 STAP성능평가기준프로그람의 속도

같은 APT는 8Mflop/s의 직렬속도, 128파라곤마디에 대하여 490Mflop/s에 도달하였다.

그림 11-13 ㄴ)에서 HO-PD프로그람의 속도결과가 제시되였다.

지난 시기에는 Maui SP2에서 256개의 처리기를 리용했을 때 23Gflop/s의 속도를 관측하였다.

HO-PD는 10.99Gflop/s의 작업량과 61.44MB의 매우 작은 입력자료립방체를 가진 계산상 매우 효과적인 프로그람이다. 그 통신비용은 그 계산상 작업량에 비하여 상대적으로 더 작다. 64개의 처리기로 구성된 경우에 SP2 T3D, 파라곤은 각각 5.2, 2.1 그리고 1.4Gflop/s의 속도에 도달하였다.

128개 마디의 경우에 SP2은 그림 11-13 ㄴ)에서 보다싶이 최대속도 9.8Gflop/s에 도달하였다.

128마디파라곤은 HO-PD를 실행할 때 2.0Gflop/s의 속도에 도달하였다. GEN-FET프로그람의 속도결과가 그림 11-13ㄷ)에 제시되였다.

세개의 MPP들의 순위는 변하지 않는다.

128개 마디의 기계크기에 대하여 SP2은 2.5Gflop/s, 파라곤은 640Mflop/s에 도달하였다.

FFT프로그람은 98MB의 큰 자료립방체를 리용한다. 16개 혹은 그이하 개수의 마디를 가지는 소규모형태에 대하여 파라곤과 T3D가 둘 다 속도감소를 보여 주었다.

EL-Stag프로그람의 속도결과를 그림 11-13 ㄹ)에 제시하였다.

이 프로그람은 그것이 110.6MB를 초과하는 집체적통보조종을 요구하므로 큰 통신부하를 체험하였다.

병렬실행시간의 분석

병목현상을 례증하기 위하여 세개의 MPP들에서의 HO-PD실행시간의 약화를 8개이하의 마디를 가지는 더 소규모적인 기계에 대하여 그림 11-4 ㄱ)에 제시하였다.

그림 11-14 ㄴ)는 32-128개까지의 마디를 가지는 기계에 대하여 제시되였다.

매 기계크기그룹에서 왼쪽 막대기는 SP2, 가운데막대기는 파라곤, 오른쪽 막대기는 T3D에 대응하는것들이다.

디스크접근시간은 파라곤의 소규모형태에서만 론의된다.

소규모형태에서 병목현상은 디스크접근벌칙에 의하여 발생한다.

파라곤에 대하여 디스크접근시간은 전체 실행시간의 약 73%인 117s이다.

병목현상은 중간 혹은 큰 개수의 마디를 가진 모든 기계들에 대하여 묶음을 이루고 있다. 분명히 단일마디기계에서는 통신비용이 없다.

집중통신들을 위한 디스크접근시간은 계산파정에서의 접근시간보다 더 길다. 디스크접근은 제한된 국부기억기를 가진 기계의 성능에서 병목현상이다.

디스크접근시간을 제외하면 묶음형성은 32 또는 그이상 마디를 가진 모든 MPP들에서 STAP프로그람을 실행하는데 대부분시간을 소비한다.

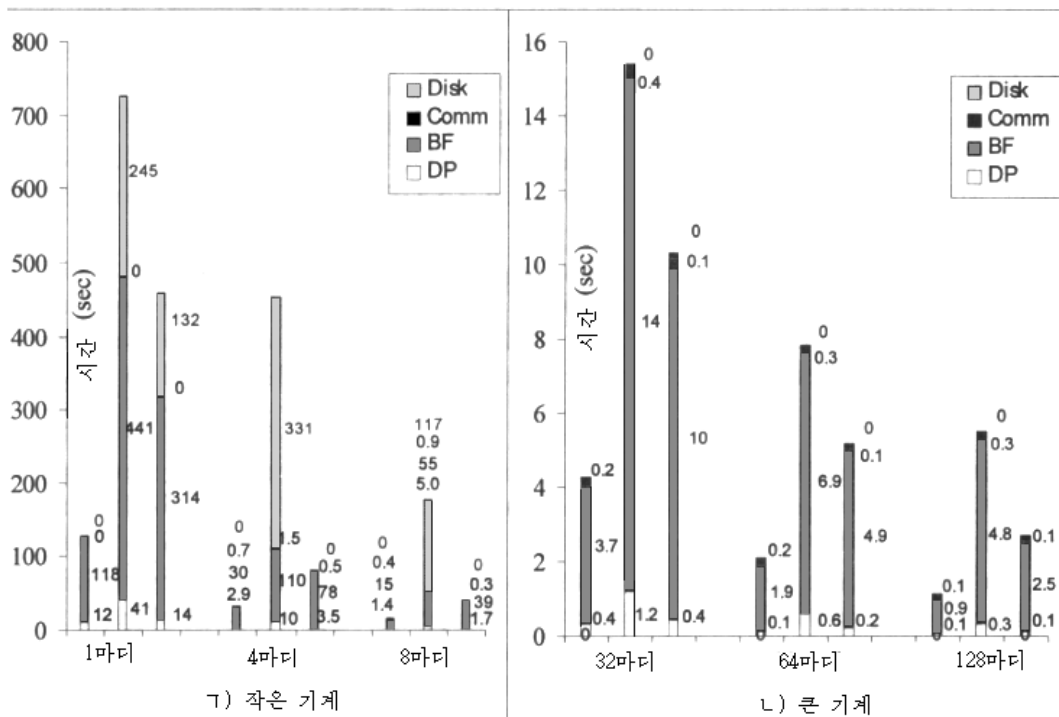



그림 11-14. 세가지 MPP 에서 HO-PD 실행시간분석

우리는 256개이상의 마디들을 가진 초대규모기계에서의 자료는 측정하지 않았지만 경향성은 변하지 않을것으로 본다. 마디처리기들의 계산능력은 묶음형성조작수행시간을 결정한다.

통신 대 계산비

통신 대 계산비(CCR)는 STAP성능평가실험들에서 측정된 총 계산시간으로 총 통신시간을 나누는 방법으로 정의한다.

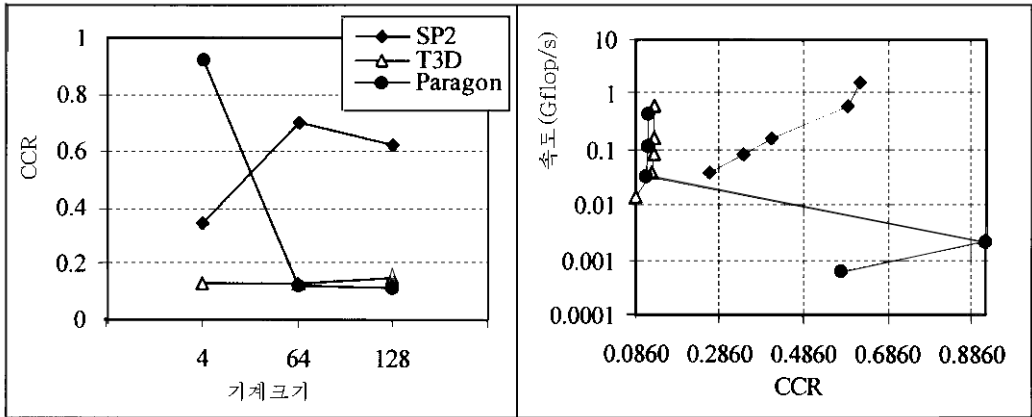



그림 11-15. MPP 속도성능에서 CCR값의 효과

CCR는 통신할 필요가 없는 한마디응용구획에서는 0과 같다. CCR의 더 낮은 값은 통신기구MPP의 행계산능력보다 상대적으로 더 효과적이라는것을 의미한다.

그림 11-15에서 기계크기에 따르는 CCR의 변화는 EL-Stag프로그람에 대하여 제시되였다.

T3D는 세개의 MPP들가운데서 가장 낮고 안정한 CCR를 가진다.

T3D는 세가지 기계들가운데서 가장 효과적인 통신기구로 장비되여 있다. SP2은 파라곤과 T3D보다 더 큰 CCR를 가지고 있다.

SP2과 T3D는 둘 다 비교할만한 통신시간을 가지고 있는데 SP2에서의 통신시간은 T3D에 비하여 절반이하이다. SP2에서의 CCR가 큰것은 주로 SP2마디의 고속처리기들에 기인된다.

SP2의 전반적인 성능이 세개의 MPP들가운데서 가장 높지만 SP2을 위한 더 좋은 통신부분체계를 얻자면 T3D나 파라곤에서보다 성능을 더 현저하게 개선하여야 할것이다. 그림 11-15 ㄴ)에서 EL-Stag프로그람에 대한 CCR값에 따르는 속도변화를 볼수 있다.

속도곡선은 왼쪽 낮은 모서리로부터 시작하여 오른쪽 웃방향으로 가면서 증가한다. CCR에서 곡선이 더 증가되면 속도감소를 일으킨다. 일반적으로 CCR값이 작으면 일부성능을 조절하여 속도를 더 높일수 있다.

큰 CCR에 대하여서는 병렬알고리듬이 다시 설정되여야 한다.

낮은 CCR는 통보문넘기기함수를 초기화하는데서 낮은 쏘프트웨어, 통보를 합성 또는 분배하는데서 통보협조처리기속도, 호상련결망에 리용된 중계접속알고리듬에 기인된다.

STAP와 같은 대규모신호처리응용은 매마디에서 소비된 계산시간보다 훨씬 더 큰 지연시간의 조잡한 병렬성만을 리용할수 있다. 그림 11.5 ㄱ)는 T3D가 3개의 MPP들 가운데서 가장 작은 CCR를 가진다는것을 보여 준다.

체계리용률

모든 세가지 MPP들의 체계리용률대역을 모든 STAP프로그람들에 대하여 그림 11-6에 제시하였다.

SP2은 최대 총사용률의 약 30%에 도달하였다.

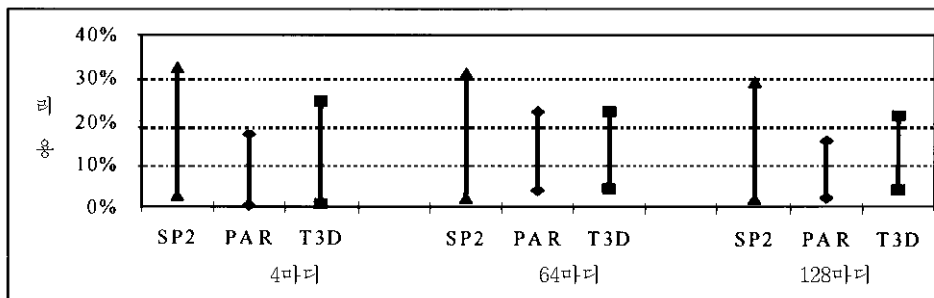



그림 11-16. 각각 4, 64, 128개의 마디들을 가지는 기계에서 모든 STAP프로그람들이 실행에 대한 SP2, 파라곤(PAR), T3D의 체계리용률

파라곤(그림 11-16에서 PAR)은 세가지 기계가운데서 가장 낮은 사용대역을 가진다. 기계크기가 128일 때 16%의 사용률밖에 얻지 못하였다. 일반적으로 체계의 리용률은 대부분 병렬응용프로그람들에서 기계크기증가에 따라 감소한다.

11. 6. 4. MPP의 구성방식상의미

MPP성능은 많은 하드웨어와 쏘프트지원체계로써 특징지을수 있다.

우리는 STAP응용프로그람에 관하여 매 MPP의 구성방식적우결함을 아래와 같이 규정한다.

우리는 마디구성방식, 절환형호상련결망, 병렬 I/O부분체계, 쏘프트환경, MPI집중통신들에 대하여 실험하였다.

일부 MPP의 조작파라메터들을 표 11-12에 제시하였다.

표 11-12 평가된 세가지 MPP들의 조작파라메터들

구성방식특징	IBM SP2	Cray T3D	Intel Paragon
리용된 구성과 기계측	MHPCC에서 리용된 400개 마디가운데서 256개 마디 리용	Cray Eagan Center에서 리용되는 64 PEs	San Diego Supercomputer Center에서 리용되는 128개마디
점대점지원시간과 대역너비	39 μs, 35MB/s	2 μs, 150MB/s	30 μs, 175MB/s
64개 마디에 대한 전체교환의 집합대역너비	0.818GB/s	1.745GB/s	0.879GB/s
MPI실행	MPICH	CRI/EPCCMPI	MPICH

마디구성방식

마디구성방식은 총체적인 체계성능에 큰 영향을 준다.

세가지 MPP들가운데서 SP2은 거의 모든 STAP성능평가실험프로그람들에 대하여 속도와 리용측면에서 가장 훌륭하게 동작한다.

256개의 SP2마디들에 대하여 HO-PD프로그람을 실행한 결과 가장 높은 속도가 19.7Gflop/s까지 달하였다. 이것은 주로 최대 256Mflop/s의 속도와 POWER2극소형처리기를 위하여 설계된 우수한 최량콤파일러에 기인된다.

Alpha 21064는 높은 박자속도를 가지고 있지만 낮은 ILP를 리용하였다. Alpha는 이런 리유로 하여 POWER2에 의하여 압도되였다. 64개 마디경우에 SP2, T3D, 파라곤의 상대속도는 각각 5 : 2 : 1 Gflop/s이다.

이 비률은 세개 마디처리기에 대한 상대적최대속도 5 : 3 : 2에 매우 가깝다.

SP2의 또 다른 우점은 매 T3D와 파라곤마디에서의 기억기보다 큰 마디기억기를 가지는것이다.

실례로 싼디아고 초고속콤퓨터쎈터에 있는 파라곤마디들의 대다수는 불과 16MB의 국부기억을 가진다.

이 마디에 큰 자료모임을 적재하면 많은 디스크교체를 일으켜 성능을 크게 떨어 뜨렸다.

STAP실험은 우리에게 보통의 MPP마디는 적어도 128MB의 기억기를 가져야 과학과 신호처리에서 흔히 만나는 대규모자료모임을 적재시킬수 있다는것을 보여 주었다.

탐지기신호처리와 같은 실시간응용에서는 매우 큰 접근지연시간때문에 국부디스크를 전혀 리용하지 않을수 있다.

한편 세가지 MPP가 다 자기의 I/O나 관문마디에서 공유디스크배렬을 50GB까지 증대시켜 대규모자료기지를 응용하기 위하여 리용할수 있다.

절환형망의 확대가능성

통신은 MPP에서 비용이 더 지출된다. 망교통과 마디립도에 의존하는 CCR와 마찬가지로 통신성능은 절환형망지연시간, 대역너비와 일정하게 관련되여 있다.

T3D에서 점대점통보문넘기기는 가장 작은 지연시간 2μs를 제공한다.

SP2과 파라곤은 40μs이하의 지연시간을 가진다. 세가지 절환형망위상구조가운데서 4536개 마디의 Intel/Sania ASCI Option Red기계[347]의 원본개발에서 증명된것처럼 파라곤 2D메쉬가 가장 높은 크기확대가능성을 보여 주었다.

다음으로 좋은 점은 1024마디까지 확대가능한 3D원환체망이다.

SP2의 고성능절환기[592]는 Cornoll리론 쎈터[168]에서 전용형태(Custom-ordered configuration)로 512개 마디까지 확장시켰다.

병렬성I/O

우리의 MPP체계들에서는 세가지 서로 다른 I/O구성방식들이 리용되였다. 대규모 파라곤에서는 마디들이 512개의 계산마디와 16개의 I/O마디로 분할되여 있다. 파일 I/O는 한개 또는 그이상의 I/O마디를 통하여 보장된다.

이 마디들은 보통 2D메쉬의 바깔렬에 배렬된다. 기계에서의 조작체계에서는 매 I/O마디도 4.8−GB RAID−3디스크배렬에 접속되였다. Intel의 병렬파일체계(PFS)는 파일에 대한 병렬접근을 제공하였다. 계산마디로부터의 모든 디스크접근은 I/O마디와의 통보문교환을 요구한다.

우리는 망이 계산마디들사이의 규칙적인 통보들을 전송하는데도 리용되므로 I/O성능이 망교통에 의하여 더 영향을 받는다는것을 발견하였다.

여러개의 계산마디가 같은 디스크에 동시에 접근하면 열점문제가 발생할수 있다. SP2은 다른 I/O구성방식을 가지고 있는데 매 마디는 국부디스크에 련결된다. 이 경우에 I/O마디와 계산마디를 구별할 필요가 없다.

SP2에서 매 계산마디는 완전한 IBM/AIX조작체계를 실행시킨다.

디스크는 매 마디에 직접 붙어 있다. I/O마디는 쏘프트웨어에 의하여 동적으로 정의된다.

PFS는 사용자가 많은 SP2마디들과 관련한 파일들을 생성해 낼수 있도록 한다.

이것은 다중자료파일들을 보존하는데서 불합리성과 관리상 부가처리를 제거한다.

T3D에서 디스크들은 host Cray C90이나 Cray−Y−MP에만 접속된다.

I/O마디는 I/O gateway를 통하여 주콤퓨터에 접속된다. 매 I/O관문은 두개의 마디로 구성되였는데 매 마디는 4M의 단어기억기(계산처리기기억기의 절반)를 가지고 있다.

한개 마디는 매 방향에서 I/O를 조종하며 체계접근과 파일접근에 리용된다.

불필요한 기억기/디스크사건들

전형적인 STAP프로그람은 한개 마디가 매 임풀스박자사이에 100MB이상의 탐지기신호자료립방체를 처리할것을 요구한다.

한개 마디의 국부기억기에 전체 자료립방체를 적재할수 없다. 우리가 사용한 세가지 MPP들가운데서 SP2, T3D, 파라곤의 한개 마디는 기껏 64MB의 국부기억을 가진다. 파라곤은 불과 한 마디당 16MB밖에 가지지 못한다. 불필요한 기억기와 디스크접근을 검토해보았다.

실례로 파라곤은 매 계산마디에서 극소핵심부조작체계를 지원한다. 핵심부와 봉사기는 6.5MB의 기억기를 차지한다.

N*통보문완충기는 또한 1MB의 기억기를 차지한다. 이렇게 되면 마디프로그람에 요구되는 공간이 제시될 때 자료기억기로써는 8MB이하밖에 남지 않는다. 디스크절체(교환)지연시간은 대규모문제를 풀기 위하여 소규모 MPP구성을 리용하는 경우에 흔히 발생한다.

문제는 국부기억기가 전체 마디프로그람과 자료부분립방체를 유지하는데 불충분한것과 관련되여 있다.

I/O는 MPP응용프로그람에서 기본병목현상의 주요원인이다[103].

불필요한 기억기/디스크사건들은 분명히 성능효과를 떨군다는것을 립증하였다.

소규모MPP가 실제적으로 리용가능하자면 마디기억기를 256MB 혹은 그이상으로 확대시키는것이 필요하다.

집중통신의 최대화와 대규모통신완충기리용은 MPP를 I/O성능측면에서 개선한것이다.

OS와 프로그람개발환경

Cray T3D는 UNICOS MAX조작체계와 쌍을 이룬다.

UNICOS기능들은 PE마디들에서의 극소핵심부 OS와 Y-MP 혹은 C90 host에서의 UNICOS로 분할된 분산형OS이다.

극소핵심부가 설계됨으로써 PE에서 실행되는 OS쏘프트웨어의 량이 최소화된다. 따라서 쏘프트웨어가 감소되여 국부기억기는 응용자료기억과 프로쎄스를 위하여 개방된다.

쏘프트웨어환경의 성능특징에서 T3D는 가장 작은 고성능신호처리용으로 적합한 성질들인 가장 작은 실행시간편차와 예비효과를 가진다.

고속동기화와 편성일람표는 안정한 계산을 위한 주요인자이다.

T3D에서 SHMEM서고는 짧은 지연시간의 원격 <읽기>/<쓰기>조작을 보장한다.

게다가 원격목표주소의 단어내용을 자동적으로 읽기 위하여 원자적인 교환능력이 보장된다. 파라곤은 매 계산마디에 극소핵심부조작체계를 지원한다.

핵심부와 봉사프로그람은 매 마디에서 6.5MB의 주기억을 차지한다.

NX통보문완충기는 또 다른 1MB의 주기억공간을 소비한다.

SP2에서 완성된 AIX는 매 SP2마디가 표준적인 단독워크스테이션으로서 동작한다.

국부디스크는 대규모자료모임이다. 프로그람적재에 리용할수 있다. 그것은 통보문이 큰 경우에 통보문완충기로서 리용될수 있다. 필연적으로 분산형OS설계는 병렬프로그람작성환경에 영향을 준다.

우리가 진행한 STAP실험에 의하면 SP2이 사용자한테 가장 편리하다.

이 실험에서는 병렬코드개발작업, 순조로운 작업, 오유형대규모입력자료와 워크스테이션들을 조종할 때 간단한 지원수법을 고찰하였다.

T3D에서의 가상주소지원의 결함은 프로그람번역에서 문제점을 야기시키는것이다.

지어 가상주소지원에 큰 기억기가 요구된다는 사정으로 하여 우리의 일부 코드가 번역될수 없었다는것이다.

세가지 MPP는 모두 유닉스에 기초한 조작체계들로 실행된다. 그런데 실시간적OS 로는 지원할수 없다. 이것은 신호체계나 거래처리와 같은 실시간응용에서 정밀측정할 때 몇가지 문제점들을 야기시켰다. 그 문제점은 서로 다른 리용자들이 자원을 놓고 서로 다투는 간섭현상으로부터 발생한다.

11. 7. 참고문헌주해와 련습문제

Congarra 등의 {212}는 500개의 최고급초대형콤퓨터들을 분류하였다. 고성능콤퓨터 시초를 [233]에서 렬거하였다.

이 장에서는 비록 그것이 상세한것은 아니라 할지라도 많은 초대형콤퓨터들과 지난 시기의 MPP들을 개괄하였다.

Hwang의 [324]에서 초대형콤퓨터에서의 병렬처리를 평가하였다.

Caray는 초병렬계산에 대한 책 [128]을 편찬하였다.

초대형콤퓨터를 위한 병렬처리와 AI기계들은 Hwang과 DeGroot의 [330]에서 취급하였다.

Bell의 [67]은 초고속콤퓨터를 평가하였다.

Cray T3D는 Adams의 [3], Koeninger들의 [371] 그리고 Cray가 발표한 [174]들에서 서술하였다.

T3D의 경험적결과는 Arpaci들의 [40]에서 찾아 볼수 있다. Dunigan은 Intel파라곤에서 Beta시험자료들을 공개하였다[222].

Hwang들의 [332]는 MPI성능을 평가하였다. Wang들의 [634]은 T3D, SP2 그리고 파라곤우에서의 STAP성능을 발표하였다.

Intel/Sandis ASCI Option Red기계는 [347]에서 서술되였으며 Mattson들의 [433]에도 서술되였다.

T3E는 [540], [541]에서와 Cray의 Web페지 [125]에서 서술되였다.

GigagRing은 [539]에서 서술되였다. Cedar계획은 Kuck들의 [382]에서 발표되였다.

NYU Ultra콤퓨터는 Gottlieb들의 [276]에서 서술되였다.

모쟈이크는 Seitz[546]에서 주었다.

연구용다중스레드화된 MPP인 Tera는 Alverson들의 [27]에서 서술되였다.

TC 2000 MPP는 BBN기술지도서 [65]에서 서술되였다.

MITJ기계들과 Dally들의 [186]에서 서술되였다.

련결기계는 Hillis의 [309]에서, CM5는 [612]에서 각각 서술되였다.

HEP는 KOWalik의 [376]과 Smith의 [570]에서 서술되였다.

Titan초대형콤퓨터는 Siewiorek와 Koopman의 [557]에서 주었다.

Gockney는 [313], [318]에서 성능평가실험과 MPP들을 위한 성능계량화를 연구하였다.

병렬계산모형 SPMD는 Daniel들의 [187]에서 특징 지었다.

Anderson과 Lam의 [35]는 확대가능한 체계의 대역적최량화기술을 고찰하였다.

CM5와 T3D에서의 통보를 위한 구성방식적지원은 Karamchetti와 chien의 [360]에서 주었다.

Frye들의 [256]은 MPP들을 리용할 때 외부리용자대면부를 고찰하였다.

STAP성능평가실험를 위한 확대알고리듬들을 Bhat와 prasanna의 [79]에서 제기하였다.

Gunther는 MPP가동환경우에서의 OLTP를 연구[284]하였다.

Horst는 [320]에서 MPP를 평가하였다.

Blevins의 저자들은 MPP를 위한 그리고 Hupairoj와 Ni의 [472]에서 서술하였다.

Detaflops계산은 Sterling들의 [587]에서 BLITZEN구성방식을 제기하였다[90].

우리는 또한 [655]-[657]에서 MPP성능을 평가하였다.

병렬I/O의 문제는 miller와 Katz의 [447], Henderson들의 [244], Bordawekar들의 [103], 그리고 Del Rosavia와 chaudhary의 [201]에서 취급하였다.

집중통신은 MCKINLEY들의 [439]에서 완성하였다.

문 제

문제 11.1. MPP는 Tflop/s의 속도를 가지는 수천개의 처리기들, GB의 수백개 기억기들, 그리고 많은 TB의 디스크기억까지 확장할수 있다.

(1) 이상에서 언급한 MPP에서의 목표를 달성하기 위하여 요구되는 처리기구성방식의 두가지 특징을 추리하여 설명하시오. 필요하다면 주장을 지원하기 위한 실례를 리용하시오.
(2) 현대적MPP들이 고성능이 되도록 하기 위하여 필요한 4가지 체계구성방식적 특징들을 갈라 내시오.

문제 11.2. 문제 11.1에서 갈라 낸 6가지 특징들에 대하여

(1) Intel ASCI Option Red체계는 6가지 특징들을 어떻게 통합하겠는가
(2) Cray T3E체계는 6가지 특징들을 어떻게 통합하겠는가

문제 11.3. [656]에서 저자들이 제기한 MPP들에 대하여 초기의 성능예측도식을 연구하시오.

NAS묶음으로부터 성능평가실험프로그람을 선택하고 알려 진 부가처리양상을 가지는 실제적인 MPP우에서 이 성능을 예측하시오. NAS그룹이 보고한 측정된 성능에 의하여 예견되는 성능을 실증하시오.

문제 11.4. 다음과 같은 령역에서 Inel ASCI Option Red와 Cray T3E를 비교하면서 매 경우에 상대적인 우결함들을 론의하시오.

(1) 마디구성방식과 처리능력
(2) 체계호상련결과 대역
(3) 기억모형과 지연시간은폐
(4) 순수한 프로그람작성환경

문제 11.5. 그림 11−16에 제시된 자료들을 고찰하시오.

(1) 기계크기가 8~64까지 변함에 따라 왜 파라곤의 유용성이 증가하는가?
(2) SP2이 왜 다른 두 기계들보다 높은 유용성을 가지는가?

문제 11.6. 다음과 같은 간략화된 2D야꼬비완화문제에 대한 구성방식이미를 특징 짓

기 위하여 위상병렬모형을 적용하시오.

문제자료령역은 $N \times N$ 격자점들의 정방형들로 이루어 지는바 이것은 n개의 부분령역으로 분할된다. 이 부분령역은 $(N/\sqrt{n}) \times (N/\sqrt{n})$ 개 격자점들로 이루어 지며 프로쎄스를 위한 마디에 배정된다. 병렬계산은 1만번의 반복으로 이루어 진다.

매 반복에서 마디는 4개의 이웃으로부터 경계격자점들을 우선 찾고 다음에 자기의 부분령역에서 $(N/\sqrt{n}) \times (N/\sqrt{n})$ 개의 격자점들의 새로운 값을 계산한다.

다음의 SPMD코드는 병렬알고리듬을 설명한다.

표시 [·]은 그 차원에 따르는 모든 배렬원소들이 접근될수 있다는것을 의미한다.

```
M=N/sqrt(n);
For(k=0;k<10000;i++){/*10000번 반복*/

/*전송단계 : 네개의 린접으로부터 경계값을 얻는다*/
Grid[0][.]을 얻기 위하여 북쪽근방으로 경계값을 교체한다;
Grid[M+1][.]을 얻기 위하여 남쪽근방으로 경계값을 교체한다;
Gird[.][0]을 얻기 위하여 서쪽근방으로 경계값을 교체한다;
Grid[M+1][0]을 얻기 위하여 동쪽근방으로 경계값을 교체한다;
/*계산단계 : Grid의 M*M내부요소를 갱신한다*/
for(i=1;i<M+1;i++)
      for(j=1;j<M+1;j++)
            Grid[i][j]=(Grid[i−1][j]+Grid[i+1][j]+Grid[i][j−1]+Grid[i][j+1])/4;
}
```

문제의 크기가 매 차원에서 N=6384개 격자점, 프로쎄스마디의 개수는 n=4, 16, 64, 256, 1024개이며 속도가 매 프로쎄스마디당 100Mflop/s라고 가정한다.

(1) 작업량에서 류동소수점 총 연산개수는 얼마인가?
(2) 마디당 거침성은 얼마인가?
(3) 격자가 류동소수점(32−b)배렬이라고 하자. 마디당 기억기요구는 얼마이며 통신완충기에 다르는 비률은 얼마인가?
(4) 통신요구(시동시간과 대역)는 얼마인가? I/O요구는 I/O속도와 I/O자료량으로 환산할 때 얼마인가?

문제 11.7. 표 11−12의 입력량들을 고찰하자. 파라곤과 SP2를 비교하자.

(1) 최대점대점하드웨어지연시간과 대역

(2) 사용자준위(즉 MPI) 점대점하드웨어지연시간과 대역
(3) MPI 전체 교환성능
(4) 상대적인 성능들에서 몇가지 결론들을 이끌어 내시오.

문제 11.8. 매 차원에서 N=1024, N=131072의 격자점을 가지고 문제 11.7을 다시 고찰하시오.

각이한 기계들에 대하여 세가지 서로 다른 문제에 대한 결과를 비교하시오.
비교연구의 목적은 아래와 같이 두가지이다.
(1) 기계크기에서의 확대가능성을 분석한다.
(2) 문제크기에 따르는 확대가능성을 분석한다.

문제 11.9. 11.3에서 ASCI MPP자료 특히 표 11−3과 11.6에서의 입력량들을 연구하여 다음질문에 대답하시오.

(1) 왜 COTS하드웨어와 쏘프트웨어가 다음세대의 초대형콤퓨터와 MPP들의 구성을 위하여 제기되는가?
(2) 이 두가지 표에 제시된 수값들을 모두 확증하시오. 세가지 ASCI가동환경 Option이 시간프렘에 대한 요구를 만족하는가?

문제 11.10. 문제 11.9와 같은 류사한 연구를 반복하시오.

그러나 여기서는 표 11−4와 11.5에 제시된 기억기와 쏘프트웨어의 요구를 연구하는데 주목하시오.
(1) 표 11−4와 11.6의 입력량들을 비교하여 문제 11.9에서의 질문부분 ㄴ)를 반복하시오.
(2) 쏘프트웨어지원과 프로그람개발환경을 위하여 ASCI의 가동환경제작자들을 비롯하여 오늘의 콤퓨터공업이 왜 지체되는가?
(3) 공업이 ASCI요구를 만족시킬수 없는 표 11−5의 입력량들의 매개에 대하여 병렬프로그람쏘프트웨어환경에서의 발전수준에 대한 독자들의 지식에 기초하여 그 리유를 설명하시오.

제4편. 병렬프로그람작성

병렬프로그람작성에 대한 착각은 직렬워크스테이션의 프로그람화에 비하여 병렬콤퓨터들의 프로그람화를 보다 어렵게 만든다.

이것이 반드시 옳은것은 아니다. 다음의 반례들을 고찰하시오.

- 병렬기계는 대다수의 직렬응용프로그람들의 독립적이고 순차적인 실행을 보장하는데 리용된다. 리용자는 응용프로그람코드를 병렬성하지 않고도 훨씬 빠른 순차적인 기계를 얻을수 있다. 대다수 독자들은 비록 얼마 안되는 병렬기계들이지만 이런 형식이 자기의 능률을 제대로 내지 못하고 있다는데 대하여 놀랍게 생각할것이다.
- 사용자는 병렬기계에서 자료기지쏘프트웨어(실례로 Oracle)를 실행시킨다. 사용자는 워크스테이션환경에서와 같이 익숙된 Oracle기능을 볼수 있다. Oracle체계는 리용자가 볼수 있는 병렬체계를 자립적으로 리용한다. 혹은 리용자는 순차적인 C프로그람을 작성하고 지능콤파일러가 리용가능한 병렬성을 자동적으로 탐색하도록 한다.

이상에서 언급한 사실들을 실현하기 위하여 우리는 병렬프로그람작성문제를 보다 신중하게 대하여야 한다. 과학원 및 공업계통의 연구소의 연구사들은 병렬프로그람작성실현방도들을 많이 창안하였다. 그런데 이것이 명백성보다 오히려 혼잡성을 조성하였을수도 있다.

병렬프로그람작성문제는 본질상 세가지 즉 공유변수, 통보문넘기기, 자료병렬프로그람작성으로 집약되는데 이에 대하여서는 마지막장에서 취급한다.

12장에서는 여러가지 병렬프로그람작성방식들(실례로 병렬프로그람작성방법)에 대하여 론의한다. 여기서는 쏘프트웨어를 비롯하여 공유변수프로그람작성에 대하여 구체적으로 연구한다.

이 연구결과들은 병렬가동환경뿐아니라 직렬기계에도 리롭다.

13장에서는 MPI과 PVM서고를 기본도구로 리용하여 통보문넘기기형프로그람작성에 대하여 취급한다. MPI는 현재 널리 리용되고 있는 표준형으로 받아들였다. PVM은 아직 표준형으로는 되지 않았지만 매우 안전하며 널리 리용되고 있다.

이 두가지 서고는 자연상태에서 호상 지원하는것들이다.

14장에서는 포트란 90과 HPF에 중점을 두고 자료병렬수법에 대하여 취급한다.

이 두가지 언어연구수법은 MPP에도 제한되지 않는다.

사실상 현재 SMP, CC−NUMA, Cluster, MPP Cluster들은 모두 MPI, PVM, 포트란 90, HPF들을 지원할수 있다.

제１２장. 병렬파라다임과 프로그람작성모형

이 장에서는 병렬프로그람작성실현방법들을 취급한다.

우리는 우선 흔히 리용하는 병렬알고리듬적프로그람들에 대하여 론의하고 4개의 병렬프로그람작성형태들인 암시적모형, 공유변수모형, 자료병렬과 통보문넘기기모형들에 대하여 고찰한다.

이 장에서는 공유변수모형에 대하여 구체적으로 취급한다. 공유변수프로그람작성에서는 5가지 방법이 제기되여 있다. 그것은 ANSI X3H5, IEEE POSIX thread(Pthread), SGI POWER C, OPEN MP표준 그리고 저자들이 개발한 Cll 라고 하는 새로운 병렬 C언어이다.

１２. １. 파라다임들과 프로그람작성능력

병렬콤퓨터체계는 신축성이 있으면서도 리용에 편리하여야 한다.

이것은 여러가지 병렬알고리듬적파라다임들을 지원하기 위한 프로그람작성능력을 잘 보여 주어야 한다. 이때 병렬알고리듬적파라다임 (병렬프로그람작성파라다임 혹은 간단히 파라다임이라고 한다.)은 병렬체계실행을 위한 알고리듬구성방법이다.

１２. １. １. 알고리듬적인 파라다임

그림 12−1에서 요약한것처럼 일련의 병렬알고리듬적파라다임들이 창안되였다. 이 파라다임들이 실제적인 병렬프로그람에서는 흔히 여러가지 방법으로 통합된다.

참고목록체계에는 점번호들이 있는데 이것은 보다 구체적인 취급과 실례들에 대한 참고표시이다.

위상병렬

1.3.3에서 론의된 위상병렬모형은 병렬프로그람작성에서 널리 쓰이는 파라다임을 제공하고 있다. 이 파라다임은 그림 12−1 ㄱ)에 보여 주었다.

병렬프로그람은 일련의 다중단계들로 구성되여 있으며 매 다중단계는 2개의 위상으로 나누어 져 있다.

계산위상에서 다중프로쎄스 매개는 독립적으로 계산 C를 실행한다.

그이후의 호상작용위상에서 프로쎄스들은 장벽 혹은 블로크통신과 같은 하나 또는 그이상의 동기적인 호상관련조작을 실행한다.

다음으로 현재의 다중단계가 실행된다.

이 파라다임은 성긴 동기화파라다임과 아겐다파라다임으로 알려 져 있다.

일치성이 부족한 동기는 오유수정과 실행분석을 가속화하지만 두가지 난점을 가지고 있다. 그것은 호상작용이 계산과 중복되지 않는다는것과 프로쎄스파정에서의 작업부하균형을 유지하기 어렵다는것이다.

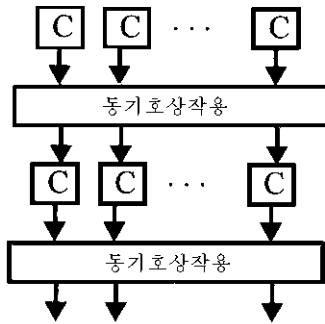
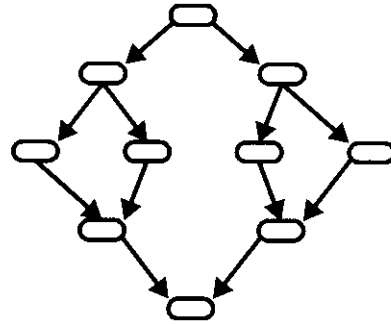
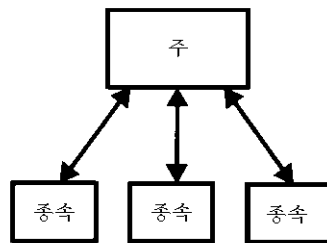
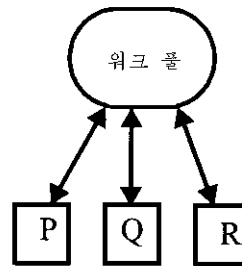



그림 12-1. 다섯가지 병렬알고리듬적파라다임들

동기와 비동기의 호상작용

위상병렬파라다임의 특수경우는 동기적인 반복파라다임이다. 여기서 다중단계들은 벡토르함수 ***x*=*f* (*x*)**를 반복계산하기 위한 다음과 같은 코드로 례증되는 한순환반복렬이다. 이때 ***x***는 *n*차원벡토르이다.

```
Parfor(i=0 ; i<n ; i++)          //매개가 다음과 같은 for순환고리로 실행되는
                                   n개 프로세스들을 생성한다.
{
    for(j=0;j<N;j++){
        x[i]=fi(x);
        barrier;
    }
 }
```

그림 12-2는 *n*=9인 경우에 코드가 어떻게 작업하는가를 보여 주고 있다.

장벽동기화에 의하여 *n*개 프로쎄스가운데 어느 하나도 *j*째 반복이 완전히 끝날 때까지 순환고리 for의 *j*+1번째 반복을 시작할수 없다.

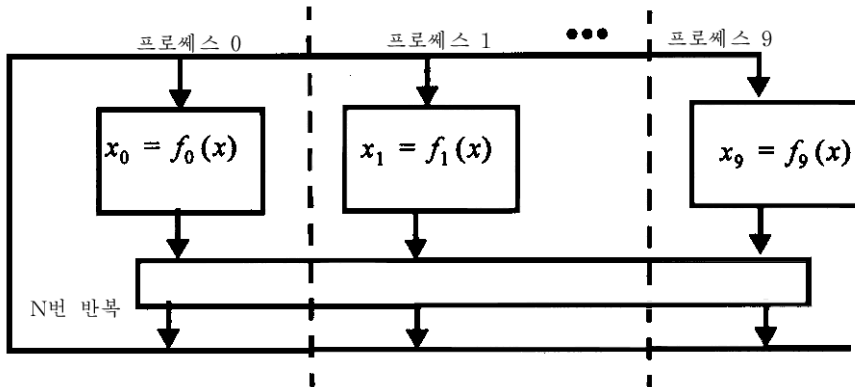



그림 12-2. 동기적반복에서 장벽동기화

반대로 비동기반복파라다임은 프로쎄스들로 하여금 나머지프로쎄스들이 따를것을 기다리지 않고 다음호상련결반복으로 넘어 갈수 있도록 한다.

이것을 아래와 같은 코드들이 확증하여 준다.

```
parfor(i:=0;i<n;i++) {
        for(j:=0;j<N;j++)
            X[i]=f(x);
}
```

이상에서 언급한 코드는 결정할수 있다. 왜냐하면 프로쎄스 j째 반복에서 X[i]가 계산될 때 리용되는 X[i-1]이 다른 프로쎄스에 의하여 j, j-1, j+1반복 등에서 계산될수 있기때문이다.

그러나 일정한 조건에서 비동기반복알고리듬은 정확한 결과에 수렴할것이며 동기반복알고리듬에 비하여 빠르다.

분할과 획득

병렬분할과 획득파라다임은 그림 12-1 ㄴ)에 보여 준바와 같이 순차적인 부분과 매우 류사하다.

부모프로쎄스에서는 자기의 과제(부하)를 여러개의 부분으로 분할하여 일련의 자식프로쎄스에 배정한다. 자식프로쎄스는 그 과제를 병렬적으로 계산하되 결과들은 부모가 현시한다. 과제의 분할과 결과의 현시공정은 재귀적으로 진행된다.

획득파라다임은 고속분류와 같은 계산에서 극히 자연스러운것이다. 결합은 부하균형을 보장하기 어려운것이다.

관흐름

관흐름파라다임은 그림 12−1 ㄷ)에 보여 준바와 같이 일련의 프로쎄스들이 가상적인 관흐름을 형성한다.

련속적인 흐름은 관흐름에 속하며 서로 다른 관흐름에서 실행되는 프로쎄스는 겹침식으로 동시에 실행된다.

프로쎄스 팜 이 파라다임은 그림 12−1 ㄹ)에서 보여 준바와 같이 **주종**파라다임으로 알려 져 있다.

주프로쎄스는 병렬프로그람의 본질적이며 순차적인 부분을 실행하여 여러개의 종속프로쎄스들이 병렬작업을 실행하도록 한다. 종속프로쎄스가 끝나면 그 정형을 주프로쎄스에 통보하며 주프로쎄스는 종속프로쎄스들에 새로운 과제를 배정한다.

이것은 주프로쎄스가 모든것을 조종하는 최고파라다임이라는것을 의미한다. 결함은 주프로쎄스가 모든 부담을 걸머지게 된다는것이다.

워크 풀 그림 12−1 ㅁ)에 보여 준바와 같이 때로는 이 파라다임이 공유변수모형으로 리용된다.

워크 풀은 대역적인 자료구조에서 실행된다.

일련의 프로쎄스들이 생성된다.

우선 풀안의 한가지 작업만이 있을수 있다.

그 어떤 빈 프로쎄스도 풀에서 한가지 작업을 맡아 실행하면서 풀안에서 0이상의 새로운 작업을 생성한다.

병렬프로그람은 워크 풀이 비면 끝난다.

이 파라다임은 작업부하가 빈 프로쎄스에 동적으로 배당되므로 부하균형을 지원한다. 그러나 다중프로쎄스들에 의하여 효률적으로 접근할수 있게 하는 워크 풀실현이 쉽지 않다. 특히 통보문넘기기모형에서 그러하다. 워크 풀은 무질서하거나 대기렬이거나 우선권대기렬일수 있다.

12.1.2. 프로그람작성능력문제

프로그람작성능력을 일반성, 이식성, 구조화의 결합으로 정의한다.

이와 같은 개념들은 아래에 서술되여 있다.

구조화

프로그람은 그것이 구성체로 이루어 져 있을 때 구조화된다.

매 구조화된 구성체는 다음의 성질들을 가진다. 즉

- 그것은 단일입력점, 단일탈퇴구성체로 고찰할수 있다.
- 서로 다른 의미론적실체들은 구성체의 문장론에서 뚜렷하게 식별된다.
- 련관된 조작이 프로그람의 다른 개소에 흩어 져 있는것이 아니라 한 구성체안에 포함되여 있다.

병렬프로그람작성에서도 사정은 마찬가지이다.

실례로 구조화된 구성체보다 구조화되지 못한 구성체에 대하여 형식의미론을 주기 어렵다[41]. 구조화된 병렬프로그람은 리해가 쉽고 다른 좋은 성질들을 가지고 있다.

실례로 구조화된 순차적프로그람들의 자료흐름분석은 순수한 문장론적방법으로 진행할수 있다.

이것은 보다 효과적인 콤파일러시간검사 및 최량화를 의미한다. 그러나 구조화되지 못한 구성체들을 허용하면 우리는 중간분석(경사흐름그라프가 축소될수 있는 경우) 또는 반복분석(일반적인 흐름그라프에 대하여)[17]과 같은 보다 복잡한 방법이 필요하다. 구조화된 병렬프로그람작성이 순차적인 프로그람작성보다 우리에게 더 필요하다.

왜냐하면 후자가 더 힘들기때문이다. 그러나 구조화방법론이 병렬프로그람작성에서는 부족점을 가진다.

구조화프로그람작성원칙은 언어구성체를 설계하는데만 적용할수 있는것이 아니라 프로그람을 설계할 때에도 적용된다.

구조화된 언어가 아닌 Fortran을 리용하여서도 구조화정도가 높은 프로그람을 작성할수 있다.

한편 보다 잘 구조화된 언어로 알려 진 C언어로도 구조화되지 못하고 리해하기 어려운 프로그람을 작성할수 있다.

일반성

병렬프로그람작성모형이 주어 진 조건에서 우리는 이 구성체들이 여러가지 프로그람들을 렬거하는데서 얼마나 유연성이 있는가에 대하여 흥미를 가질수 있다.

순차프로그람작성에 대하여 잘 알려 진 결과는 구조화정리[448]이다.

이것은 임의의 튜링계산함수가 임무, 순차적구성, 조건, while loop를 리용하여 표현할수 있게 한다.

만일 튜링계산가능성에만 주의를 돌리면 모든 병렬프로그람작성모형들은 그것이 4가지 구조화된 순차적인 구성체들을 포함하고 있는것으로 하여 일반적으로는 동등한것이다.

그러나 병렬프로그람들을 리용하게 된 기본동기는 계산속도를 높이는것이다.

때문에 우리는 일반성을 론할 때 어느 기능이 대표적인가 뿐아니라 얼마나 빨리 계산할수 있는가를 밝혀야 한다.

직관성, 일반성은 다음과 같은 의미를 넘기기하여야 한다. 즉 프로그람클라스 C와 D를 생각하자. D안의 임의의 프로그람 Q에 대하여 우리가 C안의 프로그람 P를 쓸수 있다. P와 Q가 모두 같은 의미론을 가질뿐아니라 P가 Q보다 더 좋게 혹은 Q와 마찬가지로 실행되는 그런 P라면 C는 D와 같든가 혹은 보다 더 일반적이다.

다시 말하여 일반성개념은 효과성을 포괄하며 일반적인 병렬모형은 12.1.1에서 론의된 여러가지 형식의 병렬알고리듬적파라다임들을 효과적으로 지원하여야 한다.

이식성

프로그람을 품은 적게 들이면서 한 기계에서 다른 기계에로 이식할수 있을 때 콤퓨터체계범위에서는 이것을 이식가능성이라고 한다. 이식가능성은 사용되는 언어와 목표기계에 의존되는 상대적척도이다. 실례로 이식가능성은 ①부터 ④까지의 내용을

포함한다.

① 프로그람의 알고리듬을 변경시켜야 한다.
② 알고리듬은 그대로 남아 있으나 원천코드는 변경시켜야 한다.
③ 원천코드는 그대로 남겨 두나 재편성, 재련결시켜야 한다.
④실행은 직선적으로 사용할수 있다.

또한 프로그람이 서로 다른 구성파라다임클라스의 기계들(순차적인 콤퓨터들, 공유기억기기계들, 통보넘기기기계들)사이에 호환성을 보장한다면 그것은 최대의 이식성을 가진다.

12. 1. 3. 병렬프로그람작성실례

제4장에서 고찰한 4개의 프로그람을 실례로 다시 들어 보자.

실례 12.1. π계산문제
π계산문제는 다음과 같은 반복으로 π의 근사값을 계산하는 문제이다.

$$\pi = \int_0^1 \frac{4}{(1+x^2)}dx \approx \sum_{0 \le i < N} \frac{4}{\left(1+\left(\frac{i+0.5}{N}\right)^2\right)} \times \frac{1}{N} \tag{12.1}$$

순차적인 C코드는 수값적반복에 의하여 π의 계산과정을 따라 형성된다.

```
#define N 1000000
main(){
    double local, pi=0.0,w;
    long i;
    w=1.0/N;
    for(i=0;i<N;i++){
        lical=(i+0.5)*w;
        pi=pi+4.0/(1.0+local*local);
    }
    printf("pi is %f\n",pi*w);
}/*main()*/
```

상수 N은 문제의 크기이다. N값이 크면 보다 높은 정확도가 보장되나 긴 계산시간을 요구한다.

실례 12.2. 탐지기신호처리에서 목표탐색문제

이 문제에서는 M*N형의 정렬된 점모임을 탐색공간으로 간주한다. 여기서 일부점들은 항공체계에서 비행기와 같은 목표들이다.

수값함수 IsTarget는 해당한 점이 목표인가를 시험할 때 리용된다.

매점들에 방향과 거리가 대응되여 있다.

문제는 최소거리를 가지는 목표 MaxT를 찾는데 있다.

이때 순차적프로그람에서 점 A[i][j]에는 방향 *i*와 거리 *j*가 대응되여 있다.

```
#define    MaxT     32
#define    M        2048
#deinfe    N        512
int  i,j,k=0;
float  A[M][N];
generate  the  points  A

for(j=0;j<N;j++)
    for(i=0;i<M;i++)
    {
        if(Is  Target(i,j)){
          Target[k].direction=i;
          Target[k].distance=j;
          Target[k]  .feature=A[i][i];
          K=k+1;
          If(k>MaxTarget)goto  finished;
         }
    }
finished;
```

목표탐색문제는 여러가지 파라다임들에 의하여 해결할수 있다.

병렬프로그람은 최악의 상태와 평균실행시간을 줄여야 한다.

우선 모든 목표를 찾기 위하여 IsTarget M*N번 실행한 다음 첫 MaxT목표를 찾기 위하여 그것들을 분류한다.

이와 같은 알고리듬은 최악의 상태에서 최량적일수 있으나 첫 MaxT점들이 두번째 목표로 되는 경우에는 최량일수 없다.

실례 12.3. Gaussian소거문제

Gaussian소거는 방정식 12.2와 같은 큰 선형련립방정식을 풀기 위한 유용한 직접법이다.

순차적인 Fortran코드는 다음과 같다.

```
real  A(n,n+1),x(n)
integer  i,j,k,pivot_location(1)

do  i=1,n−1
    !pivoting
    k=i
    do  j=i,n
          if(ABS(A(i,j).LT.ABS(A(i,k))then  k=j
    end  do
    swap(A(i,i:n+1),A(i−1+pivot_location(1),i:n+1))

    !triangulization
    A(i,i:n+1)=A(i,i:n+1)/A(i,j)
    A(i+1:n,  i+1:n+1)=A(i+1:n,i+1:n+1)−
    &     SPREAD(A(i,i+1:n+1),1,n−i)  *SPREAD(A(i+1:n,i),2,n−i+1)
    end  do

    !Back  substitution  follows:
    do  i=n,1,−1
      x(i)=A(i,n+1)
      A(1:i−1,n+1)=A(1:i−1,n+1)−A(1:i−1,i)*x(i)
    End  do
```

코드는 2개의 순환고리로 구성되여 있다. 첫번째 순환고리는 삼각행렬화를 실현한다. 둘째 순환고리는 거꿀순환을 실현한다. 첫 순환고리의 매 반복 i에서 절대값이 가장 큰 중심점의 위치가 먼저 구해 진다. 치환부분프로그람은 중심행을 한개의 i행으로 바꾼다.

그러면 i째 행아래의 i째 렬에 따르는 A의 모든 원소들은 령으로 된다.

실례 12.4. Jacobi완화문제

Jacobi완화법은 흔히 동기반복Gaussian에서 실현된다.

이것은 선형련립방정식 Ax=b를 풀기 위한 단순반복법이다.

즉

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{12.2}$$

우리는 A행렬이 조밀하다는것 즉 그 대다수원소들이 령이 아니라고 가정한다. Jacobi 방법은 미지벡토르 x의 초기근사로부터 시작한다. 다음으로 여러 반복걸음을 거쳐 x의 근사값들을 개량한다. 매 반복걸음에서 계산식은 다음과 같다.

$$x_i = \left[b_i - \sum_{j \neq 1} a_{ij} \times x_j \right] / a_{ii} = \left[b_i - \sum_{1 \leq j < n} a_{ij} \times x_j \right] / a_{ii} + x_i \qquad (12.3)$$

새로운 x의 값은 이것이 현재의 x값에 충분히 가까와 질 때까지 즉 $\sum_{1 \leq i < n} (\text{새로운 } x_i - x_i)^2 <$ 오차한계가 성립할 때까지 계산한다. 여기서 오차한계는 주어 진 정의 상수이다.

1 2. 2. 병렬프로그람작성모형

이 절에서는 암시적병렬프로그람작성모형과 세가지 명시적병렬프로그람작성모형(자료-병렬, 공유변수, 통보넘기기)을 소개한다.

1 2. 2. 1. 암시적병렬성

명시적병렬성은 프로그람작성자가 특수언어구성체들, 콤파일러지령들 또는 서고기능 접근을 리용하여 원천코드에 기입하는것을 의미한다.

만일 프로그람작성자가 병렬성을 명백하게 기입하지 않지만 콤파일러와 실행-시간지원체계를 자동적으로 규정할수 있게 할 때 우리는 아래에서 취급하게 되는 암시적병렬성을 얻게 된다.

병렬성콤파일러

가장 유용한 암시적병렬성수법은 순차적프로그람의 자동적인 병렬성이다. 콤파일러는 순차적인 프로그람의 원천코드우에서 의존분석을 실행하며 다음에 순차적인 코드를 본래의 병렬Fortran코드로 변환하기 위하여 한조의 프로그람번역기술을 리용한다.

이와 같은 콤파일러를 **병렬성콤파일러** 혹은 **재구조화콤파일러**라고 부른다.

순차적인 코드를 병렬성하는데서 관건적고리는 의존분석이다.

이 의존분석은 코드에서의 자료의존과 조종의존이 같다는것을 확인한다.

조작 A가 조작 B에 의존되면 A는 B이후에 실행되여야 한다. 두 조작은 그것들이 독립일 때(즉 그것들사이에 직접적이거나 간접적인 의존관계가 없을 때) 병렬로 실행된다.

의존이 있으면 프로그람번역(역시 재구조화 혹은 최량화로 알려 진)기술은 프로그람번역을 제거하든지 아니면 코드를 병렬성하는데 리용된다.

현존하는 대다수 분석 및 재구조화기술은 고리준위에 주목한다. 즉 Fortran의 순환고리 do든지 C의 고리 for에 대한 병렬성을 실행하기 위하여 어떻게 하겠는가에 주목한다.

병렬콤파일러의 효과성

의문되는 절실한 문제점은 병렬성콤파일러가 실제적인 순차적프로그람으로부터 효과적인 코드를 발생시키는데서 과연 효과적이겠는가 하는것이다. 일부 기능연구들은 그것들이 효과적이지 못할수 있다는것을 보여 주고 있다. University와 Illinois에서 발표된 Blume 등의 [91]은 12개의 순수한 성능평가기준프로그람들을 리용한 콤퓨터Cedar우에서 상업용병렬콤파일러를 받아 들인 Kap의 성능을 측정하였다.

결과들은 표 12-1의 자동양식에 제시하였다.

Cedar는 32개의 프로쎄스들로 구성된 연구용다중프로쎄스이다. 이때 매 처리기는 4단벡토르판흐름이다.

두 병렬성과 벡토르화가 충분히 리용될 때 Cedar체계는 128정도로 peak가속된다.

그러나 표 12-1에서 보다싶이 Kap콤파일러는 매우 제한된 한계에서 가속된다. 거의 절반이하의 경우에 1이하로 가속된다.

표 12-1 순수한 성능평가기준을 위한 콤파일러번역의 효과들

프로그람	자동	수동	결핍	감소	도입
ADM	0.6	10.1	9.6		
ARC2D	13.5	20.8	1.2		
BDNA	1.8	8.5	1.4	3.3	
DYFESM	2.2	11.4	2.2	2.1	
FLO52	5.5	15.3	1.0	1.1	
MDG	1.0	20.6	21.0	21.0	
MG3D	0.9	48.8	18.0	15.2	
OCEAN	0.7	16.7	3.8		8.3
QCD	0.5	20.8	8.2		
SPEC77	2.4	15.7	6.8	3.4	
TRACK	0.4	5.2	6.0		
TRFD	0.2	43.2	13.3		12.7

이 연구자들은 Kap콤파일러가 매 코드를 어떻게 병렬성시키는가를 분석하여 기능이 만족할만한 정도로 되지 못하는 기본원인이 콤파일러가 병렬성을 리용할 능력이 없는데 있다고 결론 지었다. 이것은 연구자들로 하여금 자동병렬성이 편리한 수법인가를 검토하도록 자극하였다.

그들은 12개의 프로그람을 병렬Cedar코드들로 수동번역하고 이미 번역된것들이 자동화될수 있는가를 연구하였다.

수동으로 번역한 이 코드들의 가속화성능은 표 12-1의 수동이라는 란에서 볼수 있다. 이것은 Kap로 만든 코드들에 비하여 현저히 개선되였다는것을 보여 준다. 가장 고무적인 결과로써 수동실험에서 적용된 대다수 번역을 자동화할수 있다는것을 찾았다. 또한 이 번

역은 응용프로그람에 대한 지식을 요구하지 않으며 프로그람본문으로부터 유도될수 있다.
가장 중요한 변환들가운데서 세가지가 비공개화, 병렬연역화, 귀납변수의 소거들이다.
Blume외의 저자들은 여러가지 개별적인 재구조기술의 효과들을 연구하였다.
표 12-1의 맨 오른쪽 세개의 란은 표식변환을 수동실험으로 실현하지 않을 때 속도를 늦추게 하는 요인면에서 그 결과들을 보여 준다. 실례로 ADM프로그람의 가속화는 비공개화를 적용하지 않는 경우에 다만 10.1/9.6일따름일것이다.

실례 12.5. 자료의존성제거를 위한 최량화기술

다음의 순차적do순환고리(왼쪽에 놓인)에서 명령문 Q는 명령문 P에 의존된다. 왜냐하면 Q가 P에서 계산된 변수 A의 값을 요구하기때문이다. 순환고리의 N번 반복은 이러한 의존으로 하여 병렬로 실행시킬수 없다. 비공개화기술은 A를 매 반복 i가 그 자체의 비공개복사 A(i)를 가지도록 하는 그런 렬로 만듦으로써 그 의존성을 제거한다. 비공개화된 순환고리를 오른쪽에 제시하였다.

Do i=1,N	**pardo** i=1,N
P: A=...	P: A(i)=...
Q: X(i)=A+...	Q: X(i)=A(i)+...
…	…
End do	**end pardo**

아래의 순차적인 do순환고리(왼쪽)에서 명령문 Q는 i째 반복에서의 계산합이 직전반복으로부터 얻은 합을 요구하기때문에 Q 그자체에 의존된다. 변수 A는 P에서 계산된다.
연역수법(기술)은 순환고리실행으로 얻어 지는 N개의 순차적인 더하기렬을 명시적연역함수로 전환시킴으로써 이 의존성을 제거한다.

Do i=1,N	**pardo** i=1,N
P: X(i)=...	P: X(i)=...
Q: Sum=Sum+X(i)	Q: Sum=sum_reduce(A(i))
…	…
end do	**end pardo**

귀납화는 $X(i)=X(i-1)*Y(i)$와 같은 즉 i째 반복에서 계산되는 $X(i)$가 직전반복으로부터 $X(i-1)$을 요구하는 그런 순환고리명령문이다.
자리올림을 예견한 가산기설계에 리용하는것과 같은 기술은 귀납화로 하여 생기는 의존성제거에 리용할수 있다.

자동병렬성을 초월하여

Illinois연구는 자동병렬성이 가능하다는것을 보여 주었다. 그러나 표 12-1로부터 우

리는 비록 수동적으로라도 최량화한 코드의 효률이 TRACK에 대하여 5.2/128=4%로부터 MG3D에 대하여 48.8/128=38&의 한계를 넘지 못한다는것을 알수 있다.

이 결과들은 계층적인 공유기억기를 가지는 기계인 Cedar우에서 얻은것이다. 통신지연시간이 크기때문에 MPP와 COW들은 공유기억기에 대하여 성능이 더 낮아 질수 있다.

콤파일러최량화방법을 보충하기 위하여 다른 기술이 개발되였다.

우리는 그가운데서 몇가지 중요한 기술에 대하여 론의한다.

사용자지령

사용자는 콤파일러에 많은 정보를 제공하는것으로 병렬성을 지원할수 있다. 실례로 호상작용병렬성수법은 병렬성된 콤파일러(혹은 실행시간체계)가 질문할수 있도록 하여 프로그람작성자로 하여금 병렬성처리를 안내하도록 추가적인 정보를 보장할수 있게 한다. 그러나 이 지령에서 가장 대중적인 방법은 프로그람작성자로 하여금 콤파일러지령들(역시 C에서 pragams라고 부른다.)을 원천코드에 삽입할수 있도록 한다.

이것들은 콤파일러가 재구조화작업을 더 잘 수행할수 있도록 하는 추가적인 정보를 담은 형식화된 설명문이다.

실례로 Convex Exemplar C에서의 다음과 같은 코드를 고려하여 보자. 즉

```
#pragma_CNX  loop_parallel
for(i=0;k<1000;i++) {
    S[i]=foo(B[i],C[i]);
}
```

loop_parallel 파라다임은 콤파일러로 하여금 순환고리의 몸체가 무엇인가에 무관계하게 직접 뒤따르는 순환고리가 병렬성되도록 강하게 지향시킨다. 콤파일러는 의존성을 검토하기 위하여 순환고리몸체를 분석하지 말아야 한다.

병렬성한 이후에 코드의 정확성을 담보하는것은 사용자들의 책임이다.

사용자가 함수 foo는 재입력될수 있으며 역효과가 없다는것을 알수 있다. Pragma에 의하여 보장된 이 응용프로그람지식이 없이는 콤파일러가 순환고리를 병렬성할수 없다. 왜냐하면 함수 foo가 재입력될수 있으며 역효과가 없다는것을 확정할수 없기때문이다.

실행시간병렬성

콤파일러지령의 도움을 받는다고 하여 모든 병렬성이 콤파일시간에 인정될수 있는것은 아니다. 일부가 실행시간에 나타날수 있을뿐이다. 실행시간병렬성을 실현하기 위한 기술이 제안되였다. 실례로 Stanford University에서 개발된 Jade언어에서 [517]병렬성은 콤파일러와 실행시간체계를 둘 다 포함한다.

프로그람작성자는 현존하는 순차적인 프로그람을 시동시킨다. 순차적인 프로그람을 다중과제들로 분할하고 매 과제가 자료에 어떻게 접근하겠는가를 밝혀 주는 추가적인 언어구성체들이 리용된다. 콤파일러와 실행시간체계는 자동적으로 인식하고 콤파일시간과 실행시간동안에 병렬성을 실현한다. Jade수법은 두가지 추가적인 리득을 준다. 그것은 첫째로, 병

렬성을 더 잘 인식할수 있게 하며 둘째로, 불규칙적이며 동적인 병렬성을 자동적으로 실현할수 있게 한다.

명시적병렬성

암시적병렬성은 유리한 점이 많다(12.2.3을 참고).

그리고 병렬콤파일러기술은 과학원과 공업분야들에서 활발히 연구되고 있는 수법(문제)들이다. 그러나 명시적병렬성이 필요하다고 주장하는 사람들도 적지 않다. 그들의 이러한 주장은 부분적으로는 현존 자동화수단들이 기능상 시원하지 못하다는데서 영향을 받으며 또한 A.J.Bernstein[74]이 20여년전에 얻은 리론적결과에 의하여 영향을 받는데 있다.

***Bernstein*의 정리**

명령행의 순차적프로그람에서 두가지 조작이 병렬로 실행될수 있겠는지 없겠는지에 대하여서는 아직 결정할수 없다.

이 정리가 추측해 주는것은 순차적프로그람, 잠재적인 모든 병렬성을 최대한으로 탐색할수 있는 자동화기술, 콤파일러시간, 실행시간이 없다는것이다. 이와 같은 리론적제한성을 극복하기 위하여 두가지 해결책이 제기되였다.

그것은 첫째로, 12.2.2에서 론의된바와 같이 명시적병렬성을 리용하는것이며 두번째 해결책은 명령행표현수법을 총체적으로(Bernstein의 정리는 명령형프로그람에만 적용된다는데 주의) 페지화하는것, 명령인식을 더 헐하게 하는 프로그람작성언어를 리용하는것이다. 더 론의하려면 12.2.4를 보시오.

１２．２．２．명시적병렬모형

많은 명시적병렬프로그람작성모형들이 제안되였지만 다음과 같은 세가지 모형이 주

표 12-2　　자료병렬, 통보문넘기기 그리고 공유변수프로그람작성모형의 기본특성들

특징	자료병렬	통보문넘기기	공유변수
조종흐름 (스레드화)	단일	다중	다중
동기	성긴동기	비동기	비동기
주소공간	단일	다중	다중
호상작용	암시적	명시적	명시적
자료배당	암시적 혹은 준명시적	명시적	암시적 혹은 준명시적

도적이다. 그것들은 자료병렬, 통보문넘기기와 공유변수모형들이다. 이 절에서는 간단한 실

례를 리용하여 이 세가지 모형을 요약하였으며 표 12-2에서 그 기본특징들을 개괄하였다.

자료병렬모형

이 모형은 각각 SIMD 또는 SPMD모형들에 적용된다.

기본취지는 다중계산마디에서의 서로 다른 자료모임에 대하여 동일한 지령이든지 프로그람부분을 동시에 실행시키기 위한데 있다. 본질에 있어서 자료병렬프로그람작성은 단일한 조종스레드를 가지며 초병렬성은 자료모임준위에서 실행된다. 더우기 이것은 대역적인 이름공간과 집합적자료구조병렬조작들을 가지며 성긴동기화조작들을 리용한다.

실례 12.6. 함수계산을 위한 병렬자료프로그람작성실례

다음과 같은 프로그람은 자료병렬성의 취지를 설명하기 위한 간단한 실례이다. 이 자료병렬코드는 π를 계산한다. 14장에서 HPF를 공식적으로 소개하기에 앞서 여기서 HPF와 류사한 C언어표기를 리용한다.

```
Main(){
Double  local[N],tmp[N],pi,w;
Long  I,j,t,N=1000000;
A:       w=1.0/N;
B:       forall(I=0;I<N;I++)    {
   P:          local[i]=(I−0.5)*w;
   Q:          tmp[i]=4.0/(1.0+local[i]*local[i]);
C:       pi=sum(tmp);
D:       printf(“pi  is  %f\n”,pi*w);
}/*main()*/
```

이 프로그람은 4개의 명령 A, B(두개의 부분명령 P와 Q를 가진다.), C, D를 가진다. 프로그람작성자는 이 4개의 명령문이 차례로 한개 과정으로 수행된다고 가정할수 있다.

그러나 명령문은 자료항들을 동시에 곱하기 위하여 같은 연산을 수행할수 있다. 명령문 A와 D는 바로 보통의 순차적인 명령문들이다. 명령문 C는 tmp정렬의 N개 요소들을 모두 더하는 연산을 수행하여 그 결과를 변수 pi에 넘겨 준다. 명령문 P는 동시에 N개의 오른쪽 식을 평가하며 국부정렬의 모든 4개의 원소들을 개량한다. 국부정렬의 모든 N개 원소들은 명령문 Q에서의 임의의 연산이 실행되기에 앞서 명령문 P에서 개량되여야 한다. `류사한 방법으로 tmp원소들은 어떤 C에서의 연산이 진행되기전에 명령문 Q에서 지정되여야 한다.

통보문넘기기모형

통보문넘기기모형은 다음과 같은 특성들을 가진다.

- **다중스레드** 통보문넘기기프로그람은 매개가 자기의 고유한 조종스레드를 가지

고 서로 다른 코드를 실행할수 있는 다중프로쎄스로 이루어 진다. 조종병렬성(MPMD), 자료병렬성(SPMD)이 둘 다 지원되여야 한다.

- **비동기적병렬성** 통보문넘기기프로그람의 프로쎄스는 비동기적으로 실행된다. 장벽과 통신차단 즉 실례 12.7의 MPI Reduce와 같은 특기조작들은 프로쎄스를 동기화하는데 리용된다. 구체적인 MTMD는 오늘의 통보문넘기기형체계에 리롭지 못하다는것을 주의하여야 한다.
- **독자적인 주소공간** 병렬프로그람프로쎄스들은 서로 다른 주소공간을 차지한다. 어떤 프로쎄스의 다른 변수들은 다른 처리기에 보이지 않는다. 따라서 어떤 프로쎄스는 다른 프로쎄스의 변수들을 읽거나 쓸수 없다. 프로쎄스들은 실례 12.7의 5개 MPI서고함수효과와 같은 특수한 통보문넘기기조작을 수행하는 방법으로 호상작용한다.
- **명시적호상작용** 프로그람작성자는 자료배치, 통신, 동기화, 집합화를 포함하여 호상작용문제를 모두 해결하여야 한다. 작업량배치는 보통 소유자계산규칙(owner-compute rule) 즉 자료의 어떤 부분을 소유하고 있는 프로쎄스는 그와 관련한 계산을 수행한다.
- **명시적배치** 작업량과 자료는 둘 다 사용자에 의하여 processor에 배치된다. 설계화와 코드화의 복잡성을 피하기 위하여 사용자는 흔히 SPMD프로그람작성에 단일코드방법을 리용하여 응용프로그람만을 실현한다.

실례 12.7. π함수계산을 위한 통보문넘기기코드

우리는 다음코드에서 MPI를 가진 C표기법을 리용한다. 독자들은 이 코드를 완전히 리해할 필요는 없다. 통보문넘기기조작의 상세한 내용은 13장에서 론의할것이다.

```
#define  N  1000000
    main(){
            double  local,pi,w;
            long  I,tasked,numtask;
A:          w=1.0/N;
            MPI_Init(&argc,&argv)  ;
            MPI_Comm_rank(MPI_COMM_WORLD,  &taskid);
            MPI_Comm_siae(MPI_COMM_WORLD,&numtask);
B:          for(I=tasked;I<N;I=I+numtask)        {
    P:              local=(I+0.5)*w;
    Q:              local=4.0/(1.0+local*local);
}
C:          MPI_Reduce(&local,&pi,1,MPI_Double,
                    MPI_MAX,0,MPI_COMM_WORLD);
D:          if(tasckid==0)printf(“pi  is  %f\n”,  pi*w);
            MPI_Finalize();
```

```
}/*main()*/
```

공유변수모형

공유기억기모형은 단일주소(대역적이름짓기)공간을 가진다는데서 다른 병렬모형과 류사하다. 이것은 다중스레드화적이며 비동기라는것으로 하여 통보문넘기기모형과 류사하다. 그런데 자료는 단일공유주소공간에 놓이므로 뚜렷하게 배치되지 말아야 한다. 작업량은 명백히 혹은 암암리에 배치될수 있다. 통신은 변수가 공유화된 읽기와 쓰기를 통하여 음형식으로 진행된다. 그러나 동기화는 양형식으로 진행된다.

실례 12.8. π함수계산을 위한 공유변수병렬코드

우리는 공유변수병렬프로그람작성모형의 개념을 C언어와 같은 표기법을 리용하여 설명한다.

```
#define  N  1000000
main(){
        double  local,  pi=0.0,w;
        long  I;
A:      w=1.0/N
B:      #pragma  parallel
        #pragma  shared(pi,w)
        #pragma  local(i,  local)
        {
          #pragma  pfor  iterate(i=0;N;1)
        for(I=0;i<N;i++){
     P:local=(i+0.5)*w;
     Q:local=4.0/(1.0+local*local);
        }
C:      #pragma  critical
             Pi=pi+local;
          }
D:      printf(“pi  is  %f\n”,  pi*w)   ;
}/*main()*/
```

12.2.3. 네가지 모형들의 비교

표 12.3은 리용자의 관점에서 네가지 모형들을 비교한다. 우리는 상대적우점들을 별표를 리용하여 지적한다. 네개의 별(★★★★)은 개별적인 항목들에 관하여 모형이 가장 우월하다는것을 보여 준다. 한개의 별(★)은 가장 미약하다는것을 의미한다.

표 12-3은 기술상태의 일면만을 보여 준다.

상대적우점들은 임의의 모형에 대해 반드시 고유한것은 아니다. 사실상 이 모형들의 상대적순위는 앞으로 변할수 있다.

표 12-3　　　　　4가지 병렬프로그람작성모형들의 비교

문제점		암시적	자료병렬	통보문넘기기	공유변수
가동환경의존성실례		Kap, Forge	Fortan 90, HPF,	PVM, MPI	X3H5
가동환경독립성실례			CMC*	SP2 MPL, Paragon Nx	Cray Craft, SGI Power C
병렬성론점		★★★★	★★★	★	★★
배치론점		★★★★	★★	★	★★★
호상작용 문제점	통신	★★★★	★★★	★	★★★
	동기화	★★★★	★★★★	★★	★
	집합	★★★★	★★★	★★★★	★
	비정규성	★★★★	★★	★★	★★★
의미론점 문제점	종단성	★★★★	★★★★	★★	★
	결정론	★★★★	★★★★	★★	★
	정확성	★★★★	★★★	★★	★
프로그람 작성능력 문제점	일반성	★	★★	★★★	★★★
	이식성	★★★★	★★★	★★	★
	구조성	★★★★	★★	★	★

암시적병렬성

암시적인 방법은 많은 우점을 가진다. 거대한 축적량을 가진 현재 순차적인 쏘프트웨어가 병렬콤퓨터에 재리용될수 있다. 프로그람작성자는 병렬프로그람작성 혹은 기초적인 병렬구성파라다임에 대하여 알아야 할 필요가 없다. 다만 익숙된 순차적인 언어를 리용하면 충분하다. 개발된 프로그람들은 서로 다른 병렬콤퓨터들사이에서 이식할수 있다.

명시적프로그람에 비하여 암시적프로그람이 리해하기 훨씬 더 쉽다.

사용자는 호상작용과 의미론적문제점들에 대하여 걱정할 필요가 없다.

자료병렬성

자료병렬모형은 단일주소공간을 가정하며 자료배치는 요구하지 않는다.

그러나 분산형기억기기계에서 고성능을 보장하기 위하여 HPF와 같은 자료병렬언어들은 명시적자료배치지령을 리용한다. 자료병렬프로그람은 단일스레드화되여 있으며 성긴 동기화를 가진다. 명시적동기화에 대한 요구는 제기되지 않는다. 단일스레드화는 자료병렬프로그람들이 항상 확정적이며 교착과 livelock에 무관계하다는것을 담보한다.

다른 두가지 모형은 이와 같은 우점을 가지지 못한다. 성긴동기화는 그것이 비동기

적인 반복과 워크 풀과 같은 일부 자료병렬파라다임들을 실현하기 어렵게 한다. Fortran 90 (현재 Fortran95)과 고성능 Fortran(HPF)과 같은 두가지 명백한 표준자료병렬언어가 있다. 이런 언어로 작성된 프로그람은 MPP, DSM, SMP, COW와 PVP들과 같은 넓은 범위의 콤퓨터들에 이식할수 있다. 그러나 비표준인 CM−5를 위한 C*와 같은 일부 콤퓨터 전용언어들도 있다.

통보문넘기기

사용자는 통보문넘기기형프로그람에서 처리하여야 할 작업량을 양형식으로 배치하여야 한다. 이 프로그람들은 다중스레드화되여 있으며 비동기적이다. 그리고 정확한 실행순서를 유지하기 위하여 명시적동기화를 요구한다. 그러나 이 처리기들은 자기의 독립적인 주소공간을 가진다. 공유변수의 읽기/쓰기로부터의 동기화오유는 일어 나지 않을것이다. 통보문넘기기형모형은 자료병렬모형보다 더 유연하지만 워크 풀파라다임과 대역적자료구조를 관리할 필요가 있는 응용프로그람들을 지원하는데서 여전히 약점을 가지고 있다. 널리 리용되고 있는 두가지 표준서고 PVM과 MPI들이 있다. 이것들은 모든 형식의 병렬콤퓨터에서 가상적으로 실행되며 통보문넘기기형프로그람의 이식성을 크게 돕는다. 더우기 통보문넘기기형프로그람들은 보통 성긴병렬성을 리용한다. 그것들은 본래의 공유변수모형을 가지는 기계(DSM, PVP와 SMP와 같은 다중 처리기)에서 실행할수 있다.

한편 공유변수프로그람들은 표준적으로는 다중콤퓨터에서는 직접 실행될수 없다.

공유변수

공유변수모형은 공유된 모든 자료들이 놓이는 단일주소공간에 존재한다고 가정한다. 따라서 자료배치요구는 제기되지 않는다. 공유변수프로그람들은 다중스레딩이며 비동기적이다. 그리고 프로쎄스들사이의 정확한 실행순서를 보존하기 위하여 명시적동기화를 요구한다. 더우기 처리기들은 단일주소공간을 공유한다. 공유변수의 읽기/쓰기로부터 풀기 어려운 동기화오유가 일어 나기 쉽다. 아직 널리 리용되고 있는 표준은 존재하지 않는다. SGI Power Challenge를 위하여 작성된 프로그람은 Convex Exemplar에서 직접 실행할수 없다.

SMP 혹은 DSM들을 위하여 개발된 공유변수프로그람은 MPP와 클라스터들과 같은 다중콤퓨터들에서 실행할수 없다.

공유변수(공유기억기)모형이 통보문넘기기모형보다 fine−grain병렬성을 더 잘 지원한다는것을 잘못된 생각이다. 공유변수모형은 PVP, SMP, DSM, MPP와 같은 임의의 병렬가동환경에서 실행할수 있다.

Fine−grain병렬성은 기본가동환경이 효과적인 통신 및 동기화기구를 가지면 지원될수 있다. 공유변수프로그람은 더 큰 호상작용부가처리를 생성하며 클라스터우에서 통보문넘기기형프로그람 MPP든지 지어 SMP보다 더 느리게 동작한다. 또한 리용자들은 공유변수프로그람작성이 통보문넘기기형프로그람보다 더 쉽다는것이다.

이 여론은 틀린것은 아니지만 결코 과학적실험을 통하여 확증된것은 아니다.

사실상 클라스터나 MPP들보다 SMP, PVP들을 위하여 개발된 응용프로그람들이 훨씬 더 많다. 그러나 성긴 동기화를 가지며 규칙적으로 통신패턴을 가지는 새롭고 효과적인 병렬프로그람들을 개발하는데서 공유변수수법은 통보문넘기기방법보다 반드시 쉬운것

은 아니다.

공유변수모형은 불규칙적인 병렬프로그람을 위한것이다.

이때 통보문넘기기원형을 리용하여 요구되는 호상작용을 규정하기 힘들다. 그것은 또한 통보지적자조작을 가능하게 하여 통보문넘기기형모형의 능력을 약화시킨다.

공유변수프로그람작성에서는 자료를 양형식으로 구분하여 배치하지 말아야 한다. 이것은 성능에 손해를 주기때문이다. 오유수정에서 진행되면 공유변수프로그람은 통보문넘기기형프로그람이다.

공유변수프로그람에서의 모든 처리기들은 단일주소공간에 놓이며 공유된 자료들은 차단과 림계령역들과 같은 동기구성에 의하여 보호되여야 한다. 풀기 어려운 동기화오유가 쉽게 생길수 있다. 이것은 검출하기 어렵다. 이런 문제들은 처리기들이 단일주소공간을 공유하지 않으므로 통보문넘기기형프로그람에서는 자주 나타나지 않는다.

１２．２．４．다른 병렬프로그람작성모형들

우에서 론의된 모든 병렬프로그람작성모형들은 공통적인 계산형식을 가진다. 즉 그것들은 모두 명령형모형들이다. 즉 프로그람은 동작렬을 실행하도록 체계에 명령하는 지령들을 포함한다. 기타 계산형식들은 다음과 같은것들을 포함하고 있다.

- **함수형프로그람작성** 전통적인 Von Neuman형모형은 명령형프로그람에서 그것이 어떻게 실행되는가를 해설함으로써 계산을 명백하게 밝힌다. 다시 말하여 이것은 프로그람의 조종흐름을 명백하게 한다는것(양형식으로 표현한다는것)을 의미한다. 함수형프로그람작성모형은(역시 자료흐름형식으로 알려 진) 입력자료와 계산결과사이의 함수적관계를 표현하고 있다. 콤파일러와 실행시간체계는 함수적표현으로부터 요구되는 실행렬을 생성할것이다.
- **론리형프로그람작성** 론리형프로그람작성파라다임은 일보 전진이라고 말할수 있다. 이것은 자료흐름도 함수관계도 밝히지 않는다. 대신에 무엇이 계산되여야 하는가를 밝힌다. 즉 입력과 출력사이에 론리적관계를 준다. 콤파일러와 실행시간체계는 론리적표현으로부터 알고리듬을 도출하여야 한다.
- **학습에 의한 계산** 일부 응용프로그람사용자가 알고리듬을 가지고 있지 않을뿐아니라 함수관계도 론리관계도 밝히지 않을수 있다. 례하면 사용자는 콤퓨터로 하여금 웃는 얼굴표정을 인식하도록 콤퓨터에 어떻게 명령하겠는가? 학습에 의한 계산형식은 우선 응용프로그람을 위한 학습프로그람을 구성한다. 다음에 많은 실례들이 학습알고리듬에 의하여 학습된다. 이 학습알고리듬들은 결국 과제를 수행하도록 하는 알고리듬들을 개발한다.
- **대상지향프로그람작성** 이 형식에 의하여 프로그람은 많은 대상들로 구성된다. 이때 **대상**이란 추상자료형 즉 자료구조모임(자료형)과 자료구조에 기초하여 동작하는 조작모임(방법으로 알려 진)을 의미한다. 더우기 대상들은 계승층으로 묶어 진다. 자식대상은 부모대상의 자료구조와 방법들을 계승한다.

실례 12.9. 병렬프로그람작성형식들의 비교

이제 $g=(a+b)/c$를 계산한다고 하자. 명령형식은 다음과 같은 코드 즉

```
t1=a+b;
t2=t1/c;
y=t1*t2;
```

로 작성할수 있다.

이 코드는 나누기연산에 앞서 더하기연산을 진행하도록 규정되여 있다. 함수형식은

```
y=mul(+(a,b),rec(c));/*      여기서 표시 <rec>는 거꿀수를 의미한다*/
```

와 같이 더 완화된다.

론리프로그람작성형식은 다음과 같이 더 짧게 표현된다. 즉 $y^{*}c=a+b$인 그런 y를 얻을 것이다.

이러한 비명령형식은 명령형식에 비하여 많은 우점을 가진다.

함수형을 보자. 순수한 함수형프로그람은 다음과 같다는것이 알려 져 있다.

- **일관성** 함수형프로그람은 확정적이며 교착에 무관계하게 복합적이라는것이 담보되여 있다.
- **추상성** 사용자가 병렬성, 통신, 동기화, 계획작성들을 명백하게 밝힐 필요가 없다. 사실상 많은 문제점들이 함수형프로그람에서는 제기되지 않는다.

그렇지만 명령형식은 다음과 같은 우점들도 가진다.

첫째로, 이것은 대다수 리용자들에게 익숙되여 있는 완성된 시간증명형이라는것이다. 둘째로, 이것은 Von Neuman모형이 콤퓨터하드웨어(즉Commodity Processors)와 공유됨으로써 효과적으로 실현될수 있다는것이다. 셋째로, 이것은 함수형프로그람작성이 훌륭한 성능을 가진다할지라도 많은 사람들은 여전히 명령형식을 리용할것이다. 이 명령형식은 많은 문제풀이들을 자연스럽게 표현하는데 리용되는 기초형식이다. 명령형식은 오늘의 순차적콤퓨터를 위하여서든지, 병렬콤퓨터를 위하여서든지 관계없이 우월한 프로그람작성방법이다.

이러저러한 실천적인 중요성을 고려하여 이 책에서는 명령형식의 프로그람작성만을 취급한다.

１２．３．공유기억기프로그람작성

공유기억기모형에 기초한 병렬프로그람작성은 통보문넘기기형병렬프로그람작성만큼 발전하지 못하였다.

통보문넘기기에 리용되는 MPI나 PVM과 같은 널리 리용되는 표준형이 부족한것이 그 중요한 실례이다.

현 상태에서는 공유기억기프로그람들이 다중처리기용가동환경전용언어로 리용되는 정도이다. 이런 프로그람들은 다중콤퓨터(MPP나 클라스터와 같은), 다중처리기들사이에서조차 이식성이 보장되지 않는다.

한편 SMP나 PVP들을 비롯한 모든 병렬콤퓨터에서 PVM이나 MPI를 리용하는 통보문넘기기형프로그람들이 실행될수 있다.

따라서 대부분의 현대 고성능콤퓨터들은 분명히 PVM, MPI 그리고 HPF들을 지원한다. SMP나 PVP체계는 또한 그에 고유한 공유기억기형병렬언어를 지원한다.

실천적인 표준형을 만드는것이 공유기억기형병렬프로그람작성에 대한 연구개발에서 첫번째 과제이다. 이와 관련하여 open MP가 중요한 의의를 가진다. 우리는 우선 다음과 같이 가동환경에 의존하지 않는 공유기억기형프로그람작성모형들인 X3H5, P스레드, Open MP에 대하여 론의한다.

다음에 두가지 상품화된 언어에 대하여 론의한다.

SGI Power C는 구조화된 구성체의 적은 모임을 리용하여 C언어를 공유기억기병렬언어로 확장한것이다.

Cray MPP모형은 자료병렬통보문넘기기 및 공유기억기를 위한 기능들을 통합한것이다.

우리는 끝으로 얼마간의 직교구성체(orthogonal construct)를 추가하여 통일적인 프로그람들의 개발을 촉진시키는 C//이라고 불리우는 새로운 언어를 론의한다. 통일적인 프로그람이란 구조화되고 유한이며 확정적이고 통합가능한 병렬프로그람이다.

공유기억기표준형들

여기에는 세가지 공유기억기프로그람작성표준형이 있다.

X3H5표준형은 널리 인정받지는 못하지만 여러가지 상품화된 공유기억기형언어의 설계에 영향을 미친다.

X3H5는 OpenMP표준으로 발전하였다. P스레드는 유닉스조작체계층에서 다중스레드를 위한 영향력 있는 표준형이다.

다중스레드프로그람작성은 P스레드와 많은 공통적인 특징을 가지고 있고 완성된 상업제품이므로 그것을 보여 주는 Solaris스레드를 리용한다.

１２．３．１．ANSI X3H5 공유기억기모형

X3H5공유기억기모형은 1993년에 제출된 합법적인 ANSI표준형이다. 그러나 이것은 통보문넘기기를 위한 MPI표준형인것만큼 널리 인정받지는 못하였다.

사실상 상품화된 공유기억기형체계들이 X3H5의 많은 착상을 받아 들였지만 아직

X3H5에 밀착되지는 못하였다.

우리는 12.3.1에서 X3H5의 영향을 보려고 한다. X3H5는 한가지 개념적인 표준프로그람작성모형과 C, Fortran77, Fortran90용모형의 세가지 결합을 정의한다. 우리는 기본착상에 대해서만 론의하기로 한다. 보다 상세한것은 완전명세서[30]를 참고하시오.

병렬구성체(Parallelism Constructs)

X3H5는 그림 12-3과 12.4에서 보여 주는것처럼 병렬성을 규정하기 위한 몇개의 구성체를 리용한다. X3H5프로그람은 프로그람을 실행하는데 얼마나 많은 스레드가 리용되는가를 명백히 규정하지 못한다. 프로그람은 기초스레드 혹은 주스레드라고 하는 하나의 초기스레드를 가지고 순차파라다임으로 시동된다.

```
program main                ! 프로그람은 순차방식으로 시작된다.
    A                       ! A는 기초스레드만에 의하여 실현된다.
    parallel                ! 병렬방식으로 절환
        B                   ! B는 매 조의 성원마다 복사된다
        psections           ! 병렬블로크가 시작된다.
        section
            C               ! 한 조의 성원 C가 실행
        section
            D               ! 다른 조 성원 D가 실행
        end psections       ! C와 D가 둘다 완성될 때를 대기
        psingle             ! 일시작으로 순차방식으로 절환
        E                   ! E는 한 조 성원에 의해 실행된다.
        end psingle         ! 병렬방식으로 되돌아오는 절환
        pdo i=1,6           ! pdo구축자가 출발한다.
            F(i)            ! 조성원들이 함께 F를 6번반복한다.
        end pdo no wait     ! 음형식에 장벽이 없다.
        G                   ! 더 반복되는 코드
    end parallel            ! 순차방식에로 되돌아오는 절환
    H                       ! H는 초기화과제만에 의해 실행된다.
    ...                     ! 그밖의 병렬구축자가 있을수 있다.
end
```

그림 12-3. ANSI X3H5 표준에서 병렬성구성체

병렬구성체(병렬령역이라고도 알려 진)는 닫긴 코드를 가지는 **parallel**과 **end parallel**쌍이다. 프로그람이 **parallel**과 맞다들면 그것은 령 또는 그이상의 자식스레드를 만들어 병렬파라다임으로 절환된다. 기본스레드와 그 자식스레드들은 한개 조를 이룬다. 이 조의 모든 성원들은 **end parallel**구문이 나타날 때까지의 련속적인 코드들을 병렬로 실행하도록 한다. 다음 프로그람은 순차적파라다임으로 다시 절환된다. 기초스레드만이 실행을 계속하게 된다.

병렬블로크(Parallel Blocks)

병렬구성체안에는 병렬블로크**(prections…end psections)**나 병렬순환고리**(pdo..end pdo)**나 단일프로쎄스**(psingle..end psingle)**구성체와 같은 작업공유형구성체들이 여러개 있을수 있다.

작업공유구성체가 아닌 임의의 코드들은 조의 매 성원들에 의하여 중복실행된다. 병렬블로크는 MPMD병렬성을 지정하는데 리용된다. 그것은 매개가 조의 성원에 의하여 실행되는 여러 요소-부분(작업단위)으로 이루어 진다.

병렬순환고리(Parallel Loop)

병렬순환고리는 SPMD병렬성을 지정하는데 리용된다. 이것은 모든 부분(순환렬)이 같은 코드를 가지는 병렬블로크의 특수한 형태로 간주될수 있다. 단일프로쎄스구성체는 코드가 단 하나의 조성원에 의해서 순차적으로만 실행하도록 명령한다.

그림 12-3에서의 코드렬은 그림 12-4에서 세개의 스레드를 리용하는것으로 가정하여 묘사한것이다. 초기에 코드 A를 실행하는 기초스레드 P가 있다. **Parallel**과 만나면 그것은 두 자식과제 Q와 R를 만들므로 결국 조는 3개의 성원을 가지는것으로 된다. 코드 B는 작업공유형구성체에 들어 있지 않으므로 이 세 스레드모두에 의하여 실행된다.

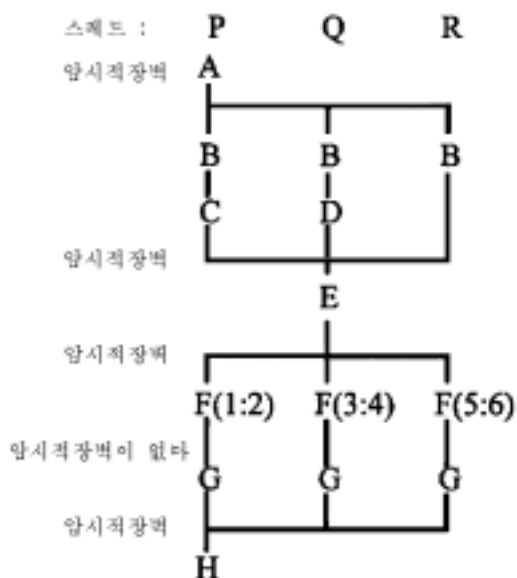



그림 12-4. 그림 12-3의 X3H5프로그람의 해설

병렬블로크**(psections)**는 조의 어떤 두 성원에 의하여 실행되는 2개의 부분 C와 D만을 가진다. 단일프로쎄스코드 E는 조성원의 임의의 하나(례하면 Q)에 의하여 실행된다. **pdo**는 조에 의해 임의의 양식으로 6번 반복실행된다. 유일한 제한조건은 매 반복과정이 1개의 조성원(스레드)에 의하여 실행되여야 한다는것이다.

실례 12.10. 작업-공유구성체

작업-공유구성체는 성원스레드사이에 부하균등화되도록 허용한다. 그림 12-3에서의

pdo에 대한 그림 12−4에서의 부하분포는 바로 가능한 하나의 대본이다. 스레드가 반복을 완료할 때 그것은 나머지 다른 반복을 실행하는데로 넘어 갈수 있다.

반복 F(1)와 F(2)이 오랜 실행시간을 가지고 나머지반복들은 짧은 시간을 요구한다고 가정하자. 그때 스레드 P와 Q는 각각 F(1)와 F(2)을 실행할것이며 스레드 R는 반복 F(3:6)을 실행할것이다.

암시적장벽

은페된 장벽으로서는 **parallel**, **end parallel**, **end psections**, **end pdo** 그리고 **end psingle**이 있다. 이 장벽은 또한 이 점에로의 모든 기억접근이 일관하도록 하는 무조건적으로 둘러 막는 조작이다. 암시적장벽이 필요 없다면 **no wait**가 추가되여야 한다. 이것은 그림 12−3의 **end pdo no wait**명령문으로 나타난다. 이제 F(1:2)가 끝난후에 스레드 P는 전체 병렬순환고리가 완성될 때까지 기다림이 없이 G를 실행하는데로 넘어 갈수 있다(그림 12−4를 참고). 병렬 및 작업공유구성체는 서로 련결될수 있다. 그때 내부병렬 혹은 작업공유구성체와 충돌하는 조의 성원이 기초스레드로 되며 그것은 새조를 형성하는 보조적인 스레드들로 분할될수 있다.

스레드호상작용

X3H5모형은 스레드호상작용에 관한 여러가지 흥미 있는 특성들을 포함한다.

- 변수는 병렬구성체내에 공유/전용속성을 가진다. 전용변수는 조의 한 성원으로 되며 그것은 조의 다른 성원들에게는 보이지 않는다. 공유변수는 조의 모든 성원들에게 속한다(그것들에 의하여 읽기/변경될수 있다.).
- X3H5는 기억기일치성문제에 주의를 돌리며 기억기의 일치성담보를 위한 방법을 제공한다. 실례는 우에서 언급된 절대적장벽이다. X3H5는 또한 뚜렷한 테두리와 둘러 막기조작을 제공한다.
- X3H5모형은 동기화대상(변수)의 4가지 형태 latch, Lock, euent, ordinal을 소개하며 초기화 및 동기화조작들과 련관된다. 표 12−4에 그 동기화대상들에 적용할수 있는 조작들과 가능한 상태들을 보여 주었다.

표 12−4 동기화대상의 상태와 조작들

대상형태	가능한 상태	대표적인 연산
크로스바	비초기화, 크로스바해제, 크로스바채우기	림계구역
차단	비초기화, 크로스바해제, 차단	검사, 차단, 비차단
사건	비초기화, clear, posted	대기, post, clear
순서	비초기화, 옹근수	순차

매 대상형은 초기화조작과 파괴조작을 가진다. 어떤 대상은 그것이 어떤 다른 조작

에 의하여 리용되기 전에 먼저 초기화되여야 한다. 크로스바, 자물쇠, 사건대상은 크로스바해제, 자물쇠해제, 지우기상태로 넘김으로써 각각 초기화된다. 오디널(ordinal)대상은 옹근수값인 2항옹근수배렬로 초기화된다. 어떤 대상은 그것을 비초기화상태로 넘김으로써 파괴될수 있다.

동기화

자물쇠와 사건동기화는 7.2.2과 류사하다. ordinal대상은 그 범위안에서 스레드를 동기화하는데 쓰인다. 실례로 우리는 ordinal대상을 스레드 T_i가 $T_{i-1},\cdots,T_1$들이 림계부분을 다 실행하기전까지는 림계부분에 들어 갈수 없게끔 지정하는데 리용할수 있다. 크로스바대상은 아래와 같은 문법을 가지는 림계령역으로 리용된다.

critical region[(latch−variable)]
critical−section−code
end critical region

이 구성체는 7.22에서 본 일반림계령역구성체와 선택적인 크로스바변수가 포함될수 있다는것을 제외하고는 비슷하다. latch를 가지지 않는 모든 림계령역은 공통적이며 암시적인 것으로, 체계가 제공하는 크로스바를 가지는것으로 간주된다. 크로스바를 리용하여 다음의 상태에서 보는것처럼 경쟁을 줄이고 병렬성을 증가시킨다.

실례 12.11. 크로스바를 리용한 호상배제

순차프로그람이 한개 처리기에서 실행될 때 10000s가 요구된다고 하자.

그것을 100개의 스레드로 분할하여 100개 짜기다중처리기에서 실행시켜 실행시간을 100s로 줄이려 한다. 문제는 이때 호상배제파라다임으로 실행되여야 할 림계부분에 있다.

스레드 T1	T2	⋯	T100
⋯	⋯	⋯	⋯
림계령역	림계령역	⋯	림계령역
⋯	⋯	⋯	⋯

초보적인 방법은 림계령역이 호상배제를 실현하는데 리용되게 하는것이다. 매 림계령역이 10s를 요구한다고 하자. 이것은 초기의 10000s라는 시간에 비하면 작은것으로 보일것이다. 그러나 호상배제로 인하여 100개의 스레드가 그 림계령역을 각기 실행해야 하므로 총 실행시간은 100*10+100=1100s이다. 속도개선은 다만 10000/1100=9만큼만 줄어 들뿐이다.

제한된 고찰에서 우리는 100개의 스레드가 50개 쌍으로 호상작용한다는것을 알았다. 즉 T_1는 T_2과만, T_3은 T_4 과만,⋯T_{99}는 T_{100}과만 호상배제되여야 한다. 좀 더 좋은 방법은 50개의 각이한 크로스바변수를 리용하게 하는것이다.

스레드 T_1	T_2	⋯	T_{100}
⋯	⋯	⋯	⋯

림계령역 (L_1)　림계령역 (L_1)　…　　림계령역 (L_{50})
…　　…　　…　　…

같은 크로스바를 가지는 림계령역만이 호상배제되여야 한다. 총 실행시간은 2*10+100=120만큼 줄어 들며 속도는 83배로 증대된다.

１２．３．２．POSIX 스레드모형(Pthread)

POSIX스레드는 IEEE표준화위원회에 의하여 제안된 공식적인 IEEE POSIX 1003. lc-1995스레드표준에 기초한것이다. 그것의 기능이나 대면부는 Solaris스레드와 류사하다. 이 절에서 우리는 IEEE표준으로부터 선택된것과 상품화된 제품들가운데서 공통적인 특성들에 주의를 돌린다.

스레드관리

스레드서고루틴은 표 12-5에 보여 준 스레드들을 관리하는데 리용된다.

Pthread_create()루틴은 프로쎄스안에서 새로운 스레드를 작성한다. 새로운 스레드는 인수 arg를 가진 myroutine을 실행한다.

새로운 스레드의 속성은 attr에 지정되든지 attr가 NULL일 때 속성의 결핍으로 간주한다. Pthread_create()가 성공되면 0을 귀환하며 thread_id안에 새로운 스레드 ID를 넣는다. 그렇지 않은 경우 오유종류를 지적하면서 오유코드가 귀환된다.

표 12-5　　pthreads에서 기초적인 스레드관리원형

함수원형	의미
int *pthread_create*(pthread_t * *thread_id*, pthread_attr_t * *attr*, void * (**myroutine*)(void *),　void * *arg*)	스레드생성
void *pthread_exit*(void * *status*)	스레드존재
int *pthread_join*(pthread_t thread, void ** *status*)	스레드결합
pthread_t *pthread_self*(void)	스레드호출귀환

pthread_exit()루틴은 접근된 스레드를 끝내게 하고 끝난 스레드와 성과적으로 결합되는 스레드에 의하여 가능한 상태를 만든다. 더우기 *pthread_exit*()는 임의의 나머지 지우기 프로쎄스함수들을 순서와 반대로 실행하는바 그후 모든 적당한 스레드특유의 파괴자가 접근된다.

Pthread_exit()에로의 암시적접근은 main()이 처음으로 접근된 뿌리스레드가 아닌 다른 스레드인 경우에 *pthread_create*()에서 지정된 *myroutine*로부터 귀환되게 한다. 귀환값은 상태(*status*)를 나타낸다. 프로쎄스에서 끝나는 스레드가 마지막스레드라면 프로쎄스는 exit()가 0상태를 가지고 접근되는것처럼 탈퇴한다. *pthread-self*()루틴은 접근된 스레드의 스레드 ID를 귀환한다.

Pthreads동기화

Pthreads동기화원형의 실례를 표 12-5에 제시하였다. 우리는 mutual-exclusion(*mutex*, 호상배제)변수와 conditional(*cond*, 조건)변수에 주의를 돌린다. 전자는 신호기(semaphore)구성체와 류사하며 후자는 사건구성체와 비슷하다. 동기화변수가 사용되기전에 그것이 창조되여야(초기화되여야) 한다는것을 강조해 둔다.

사용자가 동기화변수를 사용한 다음 그것은 지워져야 한다(기억공간비우기).

Pthread_mutex_lock()루틴은 호상배제변수가 이미 차단되지 않았다면 그것을 차단한다. 다음 그 루틴이 귀환되며 접근하는 스레드는 mutex를 얻어 그것을 자기의것으로 한다. mutex가 이미 차단되였다면 접근스레드는 mutex가 가능하게 될 때까지(해결될 때까지) 차단한다.

pthread_mutex_trylock()루틴은 test_and_set와 류사하다. 그것은 mutex를 차단하고 직접 귀환된다. 귀환값은 mutex가 이미 차단되였는가 아닌가를 가리킨다. 이 루틴은 절대로 차단되지 않는다.

표 12-6 pthreads에서 스레드호상작용원형의 실례

함수	의미
pthread_mutex_init(...)	새로운 mutex변위의 창조
pthread_mutex_destroy(...)	mutex변수무효
pthread_mutex_lock(...)	mutex변수차단
pthread_mutex_trylock(...)	mutex변수차단시도
pthread_mutex_unlock(...)	mutex변수차단해제
pthread_cond_init(...)	새로운 조건변수창조
pthread_cond_destroy(...)	조건변수무효
pthread_cond_wait(...)	조건변수에서 대기
pthread_cond_timedwait(...)	조건변수에서 시간제한까지 대기
pthread_cond_signal(...)	사건을 통보하고 한개의 대기처리를 해제
pthread_cond_broadcast(...)	사건을 통보하고 모든 대기처리를 해제

Pthread_mute_unlock()루틴은 이미 얻었던 mutex를 배제하여 준다. 다시 말하여 접근스레드는 해제조작이 성공되도록 mutex의 소유자로 되여야 한다. mutex가 해제되면 그것은 뒤따르는 다른 스레드에 의하여 얻어 지게 될수 있다.

Mutex를 기다리는 스레드가 있으면 일정짜기는 어느 스레드가 자물쇠를 얻는가를 결정한다.

어떤 스레드도 mutex를 기다리는것을 차단하지 않으면 mutex는 *pthread_mutex_lock* 혹은 *pthread_mutex_trylock*()로 접근하는 다음 스레드가 리용되게 될것이다.

이때 현행스레드가 조건변수를 기다리는것이 차단되여 mutex변수를 빈틈없이 차단하고 mutex변수를 해제한다.

대기스레드는 같은 조건변수가 *pthread_cond_signal*() 혹은 *pthread _ cond _ broadcast*()로 접근하는 다른 스레드에 의하여 신호를 받은후에만 해제되면 현행스레드는

mutex의 자물쇠를 얻는다.

pthread_cond_timedwait()는 대기시간이 한계에 도달할 때 해제된다는것을 제외하면 *pthread_cond_wait*()와 비슷하다.

Pthread_cond_signal()루틴은 조건변수를 기다리면서 차단되여 있는 한개 스레드를 해제한다. 일정짜기는 어느 스레드가 해제되는가를 결정하게 된다. *Pthread _ cond_broadcast*()루틴은 조건변수를 기다리면서 차단된 모든 스레드를 해제한다.

12. 3. 3. OpenMP 표준

OpenMP[475]는 DEC, Intel, IBM, Kuck & Associates, SGI, Portland그룹, 수값알고리듬그룹, U.S.DOEASCI프로그람 등과 같은 하드웨어 및 쏘프트웨어제작자그룹에 의하여 지원되는 공유기억기표준형이다. 그것은 Unix와 windows NT가동환경용공유기억기 API를 집체적으로 보장하는 콤파일러지령, 서고루틴 그리고 환경변수의 모임이다.

이것은 Unix와 Windows NT가동환경을 위한 공유기억기 API를 집체적으로 보장하는것이다. Fortran과 결합되는 첫 표준명세서는 1997년 10월에 나왔다[476]. C와 C++에 대한 지원이 계획되고 있다.

OpenMP의 전망

표 12-7에서 OpenMP의 전망에 기초하여 기존병렬프로그람작성표준과 OpenMP를 비교하고 있다.

우리는 이 표를 설명하는데 pthreads칸을 리용한다. OpenMP그룹은 pthreads가 기술적고성능프로쎄스가 아닌 저준위UnixSMP를 목적으로 하므로 규모변경이 필요없다. 그것은 Fortran결합밖에 가지지 못한다. pthreads는 콤파일러지령이 아니라 서고수법을 리용하므로 저준위이다.

서고수법은 콤파일러최량화를 배제한다. Pthreads는 립도가 작은 자료병렬성은 아니고 스레드병렬성만 지원한다. pthreads는 주로 Unix SMP가동환경안에서만 이식가능하고 또 지원된다(단일처리기워크스테이션을 포함하여).

표 12-7 5개 병렬프로그람작성표준의 비교

속성	X3H5	MPI	Pthreads	HPF	OpenMP
확대	아니	예	때때로	예	예
Fortran 속박	예	예	아니	예	예
C 속박	예	예	예	아니	계획화
고수준	예	아니	아니	예	예
성능지향	아니	예	아니	예	예
자료병렬성지원	예	아니	아니	예	예
Potable	예	예	예	예	예
공급자지원	아니	넓게	Unix SMP	넓게	시동
병렬성증가	예	아니	아니	아니	예

OpenMP는 X3H5개념을 대부분 계승한다. 우리는 표 12-8에서 보여 준 OpenMP와 X3H5가 차이나는 여러 중요한 특성만을 론한다.

표 12-8 X3H5와 OpenMP의 주요특성비교

속성		X3H5	OpenMP
개팔	Orphan지령	아니다	예
	서고함수와 환경변수	아니다	표준
	처리량방식	정의되지 않은	예
병렬성구성	병렬범위	병렬	병렬
	반복	PDO	DO
	비반복	PSECTION	SECTION
	단일스레드	PSINGLE	SINGLE, MASTER
자료환경	Autoscope	아니다	DEFAULT
	감소	아니다	REDUCTION
	전용초기화	아니다	FIRSTPRIVATE, COPYIN
	전용지속성	아니다	LASTPRIVATE
동기화구성	장벽	장벽	BARRIER
	방어	동기화	FLUSH
	림계구역	림계구역	CRITICAL
	작은구역	아니다	ATOMIC
	차단	아니다	충분한 가능성

이 개념은 OpenMP를 립도가 큰 병렬성을 지원하는 유연한것으로 만든다. 대부분의 순차적코드는 그림 12-5에서 보여 준 단순한 방법으로 병렬성할수 있다.

pthread는 병렬성의 증대를 잘 지원하지 못한다. 순차적인 계산프로그람이 주어 지면 그것은 사용자가 pthread들을 리용하여 병렬성하기 힘들다.

사용자는 보다 낮은 준위의 세부들을 좋아 하지 않으며 따라서 pthreads는 자연히 순환고리자료명령화를 지원하지 못한다. 따라서 사용자는 순차코드의 대부분을 변경하여야 한다.

문제 12.6에서 보는것처럼 계산을 위한 pthreads코드는 실례 12.1에서 순차적으로 서술된 그것보다는 훨씬 더 복잡하다.

OpenMP는 우에서 론의한 결함들을 해결하기 위하여 설계되였다.

```
        parameter(N = 512, NZ = 16)
        common /setup/ npoints, nzone
        dimension field(N), ispectrum(NZ)
        data npoints, nzone / N, NZ /

!$OMP PARALLEL DEFAULT(PRIVATE) SHARED(field, ispectrum)
        call initialize_field(field, ispectrum)
        call compute_field(field, ispectrum)
        call compute_spectrum(field, ispectrum)
!$OMP END PARALLEL
        call display(ispectrum)
        stop
        end

   subroutines initialize_field and compute_field are ignored

        subroutine compute_spectrum(field, ispectrum)
        common /setup/ npoints, nzone
        dimension field(npoints), ispectrum(nzone)

!$OMP DO
        do i= 1, npoints
            index = field(i)*nzone + 1
!$OMP ATOMIC
            ispectrum(index) = ispectrum(index) + i
        enddo
!$OMP END DO NOWAIT
        return
        end
```

그림 12-5. 고립지령을 현시하기 위한 OpenMP

우리는 OpenMP가 PCF와 X3H5로부터 많은것을 개선한 유망한 표준형이라고 본다. 그렇지만 그것이 널리 공인되기 위해서는 좋은 콤파일러와 실행시간환경을 개발하는 것이 중요하다.

OpenMP의 주요특징들

OpenMP는 orphan directiver개념을 결합하고 있다.

이것은 병렬구성체의 어휘론적렬에는 나타나지 않고 동적(실행)령역에 놓인다. 그림 12-5에서 설명한바와 같이 부분루틴(subroutine) compute_spectrum에서의 세가지 지령이 바로 orphan directive이다. 그것들은 주요프로그람(main program)의 병렬구성체에서 어휘론적으로 닫기지 않았지만 그 동적실행경로안에 놓인다. 우리는 한개 혹은 그이상의 병렬지령들을 리용하여 주요프로그람을 병렬성하며 기타 지령들을 리용하여 접근된

subroutine에서 실행을 조종한다. 이와 같은 파라다임으로 우리는 적은 코드변경으로 코드의 대부분묶의 병렬적실행을 가능하게 할수 있다.

이 개념은 또한 모듈화된 병렬프로그람들의 개발과 재리용을 촉진시켰다. 우리는 독자로 하여금 X3H5(orphan directive들을 지원하지 않는)로 그림 12-5를 다시 작성함으로써 orphan directive개념의 유용성을 완전히 인식할것을 권고한다(문제 12.8참고).

콤파일러지령외에도 OpenMP는 련관된 환경변수들을 가지는 실행시간서고루틴들의 모임을 보장한다. 그것들은 병렬실행환경을 조종하고 질문하는데, 일반목적자물쇠함수(general-purpose lock function)를 보장하는데, 실행파라다임을 규정하는것 등에 리용한다. 실례로 OpenMP는 처리량파라다임을 허용한다.

체계는 이때 병렬령역을 실행하는데 리용되는 스레드의 개수를 동적으로 결정한다. 이것은 한개의 응용프로그람에 소비되는 시간을 늘이는 비용에 있어서 체계의 성능을 최대화한다. X3H5와 OpenMP들은 류사한 병렬성지령들을 가진다. 다만 표기법에서 약간 차이가 있을뿐이다. OpenMP는 새로운 MASTER지령을 포함한다. 주(master)스레드만이 코드를 실행하여야 한다.

OpenMP는 X3H5보다 자료환경조종에서 보다 유연한 기능을 보장해 준다. 실례로 OpenMP는 그림 12-6에서 보여 준 PARALLEL DO지령에서 REDUCTION(+,sum)조항(clause)에 의하여 감소를 지원한다. 감소변수 sum의 전용(private)부분(copy)은 매 스레드들에 대하여 생성된다. 전용부분은 감소연산자 +에 의하여 령으로 초기화된다. 매 스레드는 전용결과들을 계산한다.

```
            program compute_pi
            integer n, i
            real*8 w, x, sum, pi, f, a
c function to integrate
            f(a) = 4.d0 / (1.d0 + a*a)
            print *, 'Enter number of intervals: '
            read *, n
c calculate the interval size
            w = 1.0d0/n
            sum = 0.0d0
!$OMP PARALLEL DO PRIVATE(x), SHARED(w), REDUCTION(+: sum)
            do i = 1, n
                x = w * (i - 0.5d0)
                sum = sum + f(x)
            enddo
!$OMP END PARALLEL DO
            pi = w * sum
            print *, 'computed pi = ', pi
            stop
            end
```

그림 12-6. OpenMP 에서 π계산

PARALLEL DO의 마지막부분에서 감소변수 sum이 그 본래의 값과 개별적계산결과를 결합한 결과와 같아 지도록 그것을 갱신한다.

+외의 다른 감소연산자들을 밝힐수 있다. 자동코드특징은 그림 12-5의 DEFAULT항목(clause)을 참고할수 있다.

DEFAULT(PRIVATE)는 다른 양형식 SHARED항목에 의하여 다시 씌여 지지 않는 한 병렬령역의 모든 변수들이 전용변수라는것을 가리킨다. 이것은 또한 모든 변수들이 공유되였다는것을 가리키는 DEFAULT(SHARED)항목이다.

자동코드화는 모든 변수들을 양형식으로 렬거할 필요가 없게 한다. 이렇게 되면 프로그람작성자의 시간이 절약되며 오유를 줄일수 있다.

OpenMP는 콤파일러로 하여금 가장 효과적인 방법을 리용하여 변수의 자동적인 갱신을 실현할수 있도록 하는 ATOMIC지령을 도입한다(그림 12-5).

이것은 2.4.1과 7.2.1에서 론의한것과 마찬가지로 림계령역, 자물쇠(1ock)와 같은 호상배제하는 구성체를 초월한다.

12. 3. 4. SGI Power C모형

SGI Power C언어는 공유변수 병렬지령(pragmas)과 서고함수를 가진 순차적인 C언어의 확장이다.

류사하게 확장된 구성체들은 포트란에 대하여서도 제공된다.

Power C설계는 그것이 구조화되여 있고 매우 간단하다는것으로 하여 효과적이다. 이것은 약간의 pragma들과 적은 수의 서고함수들뿐이다. 우리는 그것을 배우거나 리용하는데서 쉽고 많은 형태가 공학적계산문제들에서 충분히 유연하다는것을 알고 있다. 스레드관리를 위한 주요구성체들은 실례 12.12에서 해설하였다.

실례 12.12. power C스레드관리견본

다음과 같은 power C를 고찰하자. power C의 구조는 그림 12-7에 제시되였다. 병렬 parallel pragma는 다음 블로크 1 즉 "{"와 "}"로 둘러 싸여 있는 코드가 병렬블로크라는것을 지적한다. 이때 병렬블로크는 하나 또는 그이상의 스레드에 의하여 실행될수 있는것이다. 공유 pragma는 변수 *x*와 *y*들이 스레드들끼리 공유된다는것을 밝힌다.

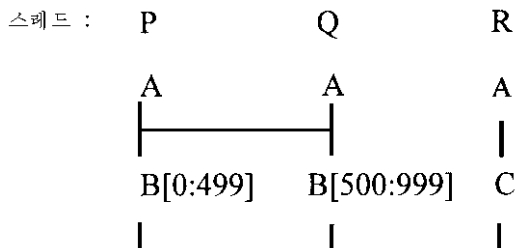



그림 12-7. power C스레드관리견본

다시 말하여 *x*와 *y*가운데서 한개의 복사만이 완전히 흠 잡을데 없는 코드이다.

어떤 스레드가 *x*를 쓸 때 기타 스레드들은 수정된 값을 보게 된다. 국부적pragma는 변수 *a*와 *b*들이 매 스레드에 대하여 국부화된다는것을 지적한다. 다시 말하여 이것은 스레드당 *a*와 *b*가운데 한개 복사품이 있다는것을 의미한다. 어떤 스레드가 *a*를 쓸 때 기타 스레드들은 변화되지 않는다.

```
#pragma parallel shared(x,y) local(a,b)// 병렬블로크

code A                                   // A는 매 스레드에 의하여 중복실행된다.
#pragma pfor iterate(i=0;1000;1)         // 작업공유구성체
{    for(i=0;i<1000;i++)code B}
#pragma independent{code C}
}
```

병렬블로크범위에서 더하기지령들은 명백한 병렬구조를 나타내는데 리용된다. 지령으로 설명을 달수 없는 임의의 코드 A와 같은것을 **국부화된 코드**라고 부른다. 국부화된 코드는 항상 모든 스레드에서 겹친다. **Pfor** pragma는 다음블로크가 작업공유블로크라는것을 의미한다. 블로크가 순환고리인것은 다행이다.

이와 같이 1000개의 반복으로 이루어 지는 작업부하는 스레드들끼리 공유한다.

두개의 스레드를 리용한다고 가정하자.

따라서 매 스레드는 프로쎄스에 대하여 500개의 반복을 가질수 있다. 작업분포는 균일하지 않아도 된다는것에 주의하시오. 한 스레드가 우연히 빨라 지면 더 많은 작업을 수행할수 있다.

독립적프로그람은 다음명령문(코드 C)이 병렬블로크안에서 다른 작업과 독립적인 스레드로 실행되여야 한다는것을 의미한다.

순차적인 코드 "A ; **for** (i=0, i<1000, i++) B; C;"로부터 유도된 이 병렬코드는 순차적인 코드와 같이 동일한 의미론을 가진다. 이것은 콤퓨터지령수법 즉 지령을 지워 버렸을 때 우리가 병렬프로그람과 꼭 같은 함수를 계산하는 유효한 순차적인 프로그람을 가지게 되는 그런 수법의 기본특징이다.

pragma들외에 power C언어는 스레드관리를 위한 얼마간의 서고함수를 보장한다.이 함수서고는 스레드, 스레드 ID 등의 대략적인 개수를 문의하는데 리용할수 있다.

동기화원형

원형들(지령들과 서고업무들)의 적은 개수는 다음과 같은것을 포함하는 스레드동기화에 리용된다.

- **림계령역** power C는 구조화된 림계령역구성체를 병렬코드안에서 서로 배제하도록 하는데 리용한다. 이것은 lock 혹은 신호기(semaphores)와 같이 구조화되지 못한 구성보다 좋은 구성체이다.
- **한개 처리기** pragma는 단 하나의 스레드로 일부 코드들을 실행할수 있다는것을

지적한다. 한개 처리기원형은 림계령역과 다르다는것에 주의하자. 여기서 요구되는 모든 스레드에 의하여 실행하여야 한다. pragma X3H5에서 표준인것과 류사하다.

- **동기화** 이것은 좋은 표준동기화원형이다.
- **모호장벽** 표준장벽에서 모든 스레드는 동기조작에 의하여 정의된 장벽점에서 동기화하여야 한다. 만일 하나의 스레드가 장벽점에 도달하지 않으면 다른 모든 스레드들은 대기하여야 하며 어떤 다른 조작도 실행할수 없다. 모호차단은 그림 12-8에 제시된것처럼 이 문제를 완화시킬수 있다.

그림 12-8. 표준차단과 모호차단의 비교

장벽은 더이상 점이 아니지만 입력과 출력원형으로 둘러 싸인 령역(zone)이라고는 말할수 있다. 스레드가 입력에 도달하면 이것은 다른 스레드를 위한 대기외에 장벽령역에서 쓸모 있는 계산코드가 실행되도록 보장할수 있다. 그런데 모든 스레드들이 입력된 령역을 가지는 동안(가질 때까지) 스레드들이 장벽령역을 내보내지 않을수 있다.

장벽령역에 쓸모 있는 계산코드를 적당히 삽입함으로써 모호장벽은 장벽대기부가처리를 크게 감소시킬수 있다.

12. 3. 5. C// : 구조화된 병렬 C 언어

우리는 새로운 병렬언어 C//(언급한 C는 병렬적이다.)를 [658]에서 제기하였다. 이것은 표준 C언어에 기초하고 있다. 표준 C언어는 프로쎄스호상작용과 병렬성을 위하여 확장한 구성체들의 작은 모임과 관련되여 있다.

C//의 핵심부에 일관한 령역(coherent region)이라고 하는 구성체가 있다. 이 령역은 구조화되고 확정적이며, 완결성을 가지며 구성적인 병렬프로그람개발을 촉진시킨다. 우리는 C//의 본질적인 특징들만을 론의하기로 한다.

일관한 병렬프로그람

병렬쏘프트웨어를 극복하기 위한 열쇠는 일관한 프로그람의 개발을 촉진시키는 병렬언어라고 본다. 여기서 일관하다는것은 다음과 같은 특성들을 념두에 둔것이다.

(1) 완결성(Terminative)

프로그람들은 항상 임의의 초기상태로부터 출발하여 실행을 끝내고 마지막상태로 들

어 간다.

병렬프로그람이 완결되지 못할 때 무한순환고리나 교착, livelock상태에 떨어 진다는 것을 상기하시오.

(2) 결정성(Determinate)

프로그람은 임의의 초기자료상태에 대하여 오직 한개의 최종자료상태만이 존재할 때 확정적이다. 즉 프로그람은 자료상태모임의 1대1넘기기이다. 결정성은 프로그람결과들의 재현을 담보하며 다른데서는 오유수정을 촉진한다.

(3) 구성적특성

프로그람의 의미론이 그자체의 구조와 구성요소들의 의미론에 의하여 유일하게 정의될 때 주어 진 총체적인 프로그람은 구성적이다. 실례로 if(C) R else T의 의미론은 그 구조 《if(…)…else …》와 구성요소 C, R, T들의 의미론에 의하여 구성적으로 정의된다.

이 구성적특성은 프로그람개발을 촉진시키며 리해를 쉽게 해준다.

(4) 구조화

프로그람이 구조화된 구성체들로 이루져 있을 때 주어 진 프로그람은 구조화된것이다. 이때 매 구조화된 구성체는 몇가지 특성들 즉

- 단일입력자료단일출력자료구성체
- 서로 다른 의미론적대상들은 문장론적구성체의 견지에서 명백히 식별된다.
- 관계연산들은 프로그람안의 서로 다른 개소들에 분산되여 있는 대신에 한개 구성체안에서 닫겨 져 있다.

와 같은 특성들을 가진다.

병렬구성체

변수와 식의 C정의와 순차적인 블로크는 대괄호쌍으로 닫겨 진 명령렬이라고 가정한다. 즉 $\{S_1\ S_2 \cdots S_n\}$

MPMD병렬은 병렬블로크리용을 밝힌것이다.

par[region_name_declarations] $\{S_1\ S_2 \cdots S_n\}$

여기서 매 성분 S_i는 다른 병렬블로크와 같은 임의의 프로쎄스일수 있다. 최량령역이름선언은 간단히 론의한다. Cll역시 속기표기, 명확한 SPMD병렬성을 위한 병렬순환고리구성체를 가진다

즉

parfor(i=e1;e2;e3)　　/*순환고리첨수변수를 가지는 순환고리머리부*/

```
[region_name_   declarations]
{S}                         /*순환고리몸체*/
```

순환고리머리부에 있는 식 $e1$, $e2$, $e3$들은 순차고리와 류사한 파라다임으로 순환고리첨수값들의 모임 $I = \{I_1, I_2, \cdots, I_n\}$을 생성하는데 리용된다. 즉

$$I = \varnothing \ ; j = \mathrm{el} \ ; \ \ \mathbf{while}\,(\mathrm{e2})\ \{I = I \cup \{i\}\ ;\ \mathrm{e3};\}$$

이다.

$n(I)$(혹은 간단히 n)가 I의 원소개수라고 하자.

병렬순환고리구성체는 $n(I)$개의 프로쎄스들을 생성한다. 이때 매 프로쎄스는 식별코드 S의 복사이며 매개는 서로 다른 첨수값을 리용한다. 병렬for순환고리는 등가적인 병렬블로크를 위한 속기표기로 볼수 있다. 일반적으로 **parfor**(e1 ; e2 ; e3) {S}는

par{{S(I1)}{S(I2)}⋯{S(In)}}와 등가이다.

실례로 병렬블로크 **par**{Process(1)⋯Process(n)}은 **parfor**(i=1;i<=n;i++) {Process(i)}와 등가이다.

여기서 $I = \{1, 2, \cdots, n\}$이다.

순환고리머리부는 임의의 순환고리몸체명령문이 실행되기에 앞서 완전한 첨수값모임 I를 생성하는것과 등가이다. I는 항상 유한이다. 한편 행렬순환고리는 **parfor**(i=1 ; ; i++) {⋯}의 경우처럼 끝나지 않을것이다.

I가 빈모임이면 병렬순환고리는 빈 명령문 ”;”과 등가이다. I는 순환고리머리부에서 단독으로 평가된다. 혼돈을 피하기 위하여 C//는 $e1$과 $e2$이 부작용이 없으며 $e3$은 순환고리첨수변수에만 영향을 줄것을 요구한다.

일관한 령역(coherent Region)

병렬블로크(혹은 병렬for순환고리)의 요소처리기들은 말하자면 론리정연한 령역들을 통하여 호상련결된다. 이러한 령역은 예약어 **core**에서부터 출발해서 선택령역이름, 선택령역조건 그리고 선택령역몸체를 서술한다.

```
core  region_name(region_condition)
{
region_bady
}
```

령역은 이름을 가질수도 있으며 가지지 않을수도 있다.

이름을 가진 임의의 령역은 다음과 같이 **par**블로크(혹은 별렬순환고리)의 앞에서 선언된 령역이름을 가져야 한다. 즉

```
par
    region region_name [particlpant_list];
{…}
```

이여야 한다.

여기서 선택부속자목록은 호상련결을 위한 령역을 리용하는 모든 프로쎄스들(령역의 부속자라고 하는)을 일일이 렬거한다. 이 목록이 정확하게 기록되지 못하였을 때 병렬블로크의 모든 요소처리기들이 주어 진 목록의 부속자들이다.

령역조건은 부작용이 없는 론리식이다. 령역조건이 성립하지 않을 때 조건표현은 항상 TRUE인것처럼 고찰한다. 이름 붙은 령역이 관계자에 의해 호출될 때 모든 령역호출에서 조건들은 식별되여야 한다.

령역몸체는 병렬명령문과 순차적명령문들의 렬로 이루어 진다. 순차적명령문은 (조건)대입명령문렬을 포함한다.

병렬부분은 다음과 같은 명령문 즉

- 배정.
- “var=op(식);”형식의 집합적배정
  여기서 op는 귀납이든지 주사조작이다. 이것들은 “sum_reduction”,”minimum_scan”들과 같은것들이다.
- “**if**(조건)배정”이든지 ”**if**(조건)집합적배정”형식의 조건적배정. 조건은 임의의 부작용을 가지지 말아야 한다.

들을 포함한다.

매 병렬토막은 단일배정규칙을 만족시켜야 한다. 이 규칙은 변수에 한개이상의 토막을 배정할수 없다는것을 말해 준다.

일관한 령역의 의미론

프로쎄스가 **core** R에 도달되면 그것은 령역 R의 관계자들도 **core** R에 도달될 때까지 대기한다.

그러면 이 령역에 대하여 령역조건은 등가이다. 조건이 FALSE와 등가이면 모든 부속과 프로쎄스들은 조건이 TRUE로 될 때까지 대기한다. 그러면 모든 부속자들이 령역몸체를 실행하기 위하여 령역을 병렬로 입력시킨다. 몸체가 실행된후에 모든 부속자들은 그 각자의 차후계산을 실행하기 위한 령역을 남겨 둔다. 이름 붙지 않은 령역의 부속자만이 그것에 접근하는 프로쎄스라는데 주의하자.

령역조건의 평가와 령역몸체의 실행은 자동적인 트랜잭션이라고 본다. 령역몸체(우리는 령역의 모든 몸체들은 단일령역으로 결합된다고 가정한다.)는 다음규칙에 따라 실행된다.

즉

- 이것들이 령역몸체에서 다중토막들일 때 그것들은 순서대로 실행된다.
- 순차적인 토막은 순차적인 블로크처럼 실행된다.
- 병렬토막에서 모든 배정들은 아래와 같은 문법을 가진 병렬배정명령을 개념적으로 구성한다고 볼수 있다

.

Variable−1,⋯,Variable−N=Expression−1,⋯,Expression−N

의미론은 모두 N개의 식들이 N개의 값으로 평가된다는것이다. 그러면 N개 값들을 N개의 대응변수들로 가정한다.

```
LOCK(S)
 Critical section
UNLOCK(S)
```

ㄱ) 차단을 리용한 호상배제

```
core (S) { S=S-1; }
   Critical section
core { S=S+1; }
```

ㄴ) C// 호상배제를 위한 코드

```
await(S);
signal(S);
clear(S);
```

ㄷ) 3개사건연산

```
core (S){}
core {S = 1;}
core {S = 0;}
```

ㄹ) C// 3개사건연산을 위한 코드

```
await (condition)
   do critical_section
```

ㅁ) 조건경계범위

```
core (condition)
 { seq critical_section }
```

ㅂ) C// 조건경계범위를 위한 코드

```
result = F&A(x, v)
```

ㅅ) 꺼내기첨가연산

```
core { result = x; x = x + v; }
```

ㅇ) C// 꺼내기첨가를 위한 코드

```
C&S(w, old, new)
int * w, old, new;
{ if (*w == old)
    {*w = new; return 1; }
  else  return 0;
}
```

ㅈ) 비교와 교체

```
core {
    seq
        result = (w ==old)? w=new,1:0;
}
```

ㅊ) C// code for "result = C&S(&w,old,new)"

```
core R {}
```

ㅋ) 장벽동기화 C//

```
core R {s = Sum_reduction(a[i]) ; }
```

ㅌ) Sum reduction in C//

그림 12−9. C//에서 일관한 령역구조를 리용한 여러가지 동기화의 실현

프로쎄스호상작용지원

일관한 령역은 만능적인 구조이다. 통신화와 집약화들은 실례 12.13과 12.14들에서 각각 보여 준다. 동기화는 어떻게 실현하는가를 그림 12−9에 제시하였다.

일반적으로 원자적령역동기화 혹은 림계령역동기화는 무조건적인 이름 붙지 않은 일관한 령역에 의하여 실현된다. 조종동기화는 이름 붙은 령역에 의하여 실현되며 자료동기화는 조건적인 일관한 령역에 의하여 실현된다. 경쟁적인 호상작용은 프로쎄스들이 공유자원에 대한 경쟁을 통하여 호상작용하는 때를 의미한다.

기타 모든 호상작용을 **협동적**이라고 부른다.

보통 협동적호상작용들은 경쟁적인 호상작용이 이름 붙지 않은 조건령역에 의하여 실현되는 동안에 이름 붙은 무조건적령역에 의하여 실현된다.

실례 12.13. C//에서 통신화를 어떻게 실현하는가

다음과 같은 병렬블로크가 교환, 방송, 집단내 방송 그리고 조건적배정을 어떻게 실행할수 있겠는가를 보여 준다. 즉 그것은

```
parfor(i=0;i<n;i++)
     region  Region1;
{…
     core  Region1
     {a[i]=d;                    //1대  n 방송 :즉 d는 a[0],…,a[n−1]로 방송된다.
     b[i]=e[f[i]];               //f[I]가 m개의 서로 다른 값과 같으면 m대 n 방송
     c[i]=c[i+1 mod n];          //자리바꿈
     if(I%2==0) left[I+1]=right[I−1]=u[i];}        //조건부적배정
  }
}
```

조건부적배정은 통신에 대한 일부 프로쎄스들을 위하여 쓸모 있다. 실례로 유일하게 번호 붙여 진 프로쎄스들만 수값우에서 언급한 령역에서 u[i]값들을 내보낸다.

완전한 토막은 병렬배정이다.

특히 “c[i]=c[i+1 **mod** n]”을 위한 병렬배정부분은

c[0], c[1], …, c[n−1] = c[1], c[2], …, c[n−1], c[0]

과 등가이다.

Regionl의 실행마지막에 정렬 C의 왼쪽 밀기연산 즉 new c[0]=old c[1], new c[1]=old c[2], …, new c[n−1]=old c[0]와 같이 연산이 수행된다.

실례 12.14. C//에서 집약화가 어떻게 실현되는가

집약적호상작용의 다른 형식은 프로쎄스들이 그 값들가운데 일부값들을 모아 놓을것

을 바라는 경우가 생성된다.

이 호상작용형은 집약화연산자(즉 귀납과 증상)를 통하여 지원된다. C//은 사용자들이 자체의 집약화연산자들을 정의할수 있게 하기 위한 기구들을 보장한다.

다음과 같은 코드는 n−차원배렬 A의 기둥도표

hist[i] (for i=0,1,⋯,)

를 계산한다.

그것은 병렬순차토막의 리용은 물론 감소연산자의 리용을 설명한다.

```
Parfor(I=0;I<n;I++)
      Region R;
{core R
         amax=max_reduction(A[i]);      /*변수 amax에 A의 최대값을 배정*/
         amin=mon_reduction(A[i]);      /*변수 amin에 A의 최소값을 배정*/
 seq
         binsize=(amax−amin)/binnumber;
 parallel
         bin[i]=1+(a[i]−amin)/binsize;
 parallel
         hist[bin[i]]=count_reduction(bin[i]);
  }
}
```

count_reduction 연산자는 매 자료값의 발생수를 계산한다.

C//언어의 현상태(Status of The C// language)

이 프로그람작성언어는 어느 한 연구쎈터에서 활발히 진행되는 연구과제이다.

C//의 수많은 실례프로그람들이 작성되고 콤파일되여 일부 SMP와 클라스터화된 병렬콤퓨터상에서 실행되였다. C//에 대한 콤파일러는 아직 개발되지 않았다. 그 효과성을 검증하자면 완전한 C//언어정의와 콤파일러들이 대단히 요구된다.

우리는 병렬프로그람작성연구집단에 확대가능한 병렬콤퓨터체계상에서 C//를 실행할수 있는 콤파일러전처리기 그리고 실시간서고들을 개발할것을 호소한다.

12. 4. 참고문헌주해와 련습문제

여러가지 책들에서 병렬알고리듬적파라다임들을 론의하였다. 이런것들가운데서 가장 최근의 결과들은 [111]에서 취급하였다. [78]에는 동기화 혹은 비동기화된 반복알고리듬들이 포함되여 있다.

[130]은 세가지 파라다임들 즉 일정조작파라다임, 결과, 전문가파라다임들을 론의하였다. [63]은 선형체계에 리용된 견본들을 론의하였다.

[60], [367], [647], [669]들은 병렬콤퓨터기술에 대한 좋은 참고서이다. [91, 92]은 병렬

콤파일러에 대한 연구결과들을 취급하였다.[292].

여기서는 stanford SUIF콤파일러가 SPEC92와 SPEC95와 같은 순차적인 성능평가기준프로그람들이 효과적으로, 병렬적으로 널리 리용된다는것을 보여 주었다. 공유변수프로그람작성에서 가장 큰 문제점은 표준형들이 없는것이다. ANSI X3H5표준형은 현재 피동적이며, X3H5공유기억기표준형은 공업적으로 실현되지 못하고 있다.

[37]로부터는 병행프로그람에 대한 좋은 결과들을 찾아 볼수 있다.

실천적으로 대다수프로그람작성자들은 적합한 프로그람작성언어(즉 GI power C언어 Crag craft) 혹은 스레드서고(즉 Solaris 스레드든지 P스레드들)가운데서 어느 하나를 리용한다.

OpenMP표준은 공유기억기다중처리파라다임을 단일화할수 있다.

P스레드는 유력한 IEEE/ANSI표준이며 [345]에서 서술하였다. 많은 상업용조작체계들은 P스레드와 류사하거나 호환성 있는 스레드모형을 실현하였다.

구체적인 스레드체계들은 [192, 193, 340, 601]에서 서술하였다. X3H5표준형은 [39]에서 취급하였다.

OpenMP표준은 [475, 476]에 서술되여 있다. 그리고 cray craft모형은 [480]에서 취급하였다. [166]은 Exemplar기계우에서의 공유기억기프로그람작성모형을 서술하였다.

SGI [547]에 의하여 pow와 C프로그람작성모형이 개발되였다. 이것은 Forran과 류사한 모형이다. 일관한 령역구성체와 C언어는 [658]의 저자들이 제기한것이다.

함수형언어를 리용한 병렬프로그람작성은 [24]에서 론의하였다.

대상지향수법들은 [15]에서 서술되였다. C++를 리용한 병렬프로그람작성은 [645]에서 주었다. 분산형계산체계를 위한 프로그람작성언어는 [58]에서 개괄하였다.

문 제

문제 12.1. 다음과 같은 프로그람작성능력개념을 해설하는데 병렬언어구성의 구체적인 실례와 계산실례들을 리용하시오.

(1) 구조성
(2) 일반성
(3) 이식성

문제 12.2. 암시적 및 명시적병렬프로그람작성모형을 비교하시오.

(1) 일반성을 제외하고 병렬조종, 자료배정, 호상작용, 의미론과 프로그람능력과 관련한 거의 모든 문헌들에서 왜 명시적모형이 암시적모형보다 우월한가를 설명하시오.
(2) 많은 병렬 및 분산형콤퓨터가동환경들에서 암시적콤파일러수법의 리용을 방

해하는 주요제한조건들을 지적하시오

문제 12.3. 아래의 파라다임들을 리용하여 두개의 옹근수배렬들의 내부결과를 계산하는 병렬코드를 쓰시오. 순차적인 코드는

sum=0; **for** (i=0; i<N; i++) sum=sum+A[i]*B[i];

(1) 단계방법
(2) 분할과 획득
(3) 관흐름
(4) 프로쎄스팜
(5) 프로쎄스머리부

문제 12.4. 병렬프로그람작성모형에 대한 다음의 문제에 대답하시오.

(1) 왜 통보문넘기기모형이 배정문제에 대하여 공유변수모형보다 나쁜가.
(2) 왜 통보문넘기기모형이 동기화, 의미론 그리고 이식가능성문제에서 공유변수모형에 비하여 우월한가
(3) 왜 암시적모형이 일반적인 모형에서 공유변수모형에 비하여 나쁜가

문제 12.5. 크기가 N=100,000와 N=10,000,000인 두 문제에 대하여 π값을 계산하기 위한 SPMD병렬프로그람을 작성하시오.

임의의 공유기억기언어와 가동환경우에서 독자의 코드를 개발하고 시험하여야 한다.

프로그람은 1, 2, 3, 4, 5, 6, 7, 8 프로쎄스들을 리용하여 서로 다른 크기의 기계에서 락관적으로 실행하여야 한다. 이 련습에는 다음과 같은것들이 포괄되여야 한다.

(1) 완성되고 잘 정리된 순차적인 병렬프로그람
(2) 가동환경과 언어, 콤파일러선택, 서고, 환경파라메터들, 실행파라다임들(묶음/호상작용, 독립/공유), 시간측정 등을 포함한 시험환경과 절차들에 대한 해설
(3) 순차적실행시간, 병렬실행시간, 속도, 가속도, 효률 및 리용을 포함하여 측정한 일람표화 및 계획표화성능수들
(4) 측정결과분석, 관찰에 주의를 돌리며 기계크기와 문제크기의 리용과 확대가능성에 대한 해설

문제 12.6. 아래의 내용들에 대한 병렬프로그람을 작성하기 위하여 스레드표기를 리용하시오.

(1) 계산문제

(2) 야끄비완화문제
(3) 가우스소거문제
(4) 목표탐색문제

이 장에서 명백히 주지 않은 구체적인 표기들을 정의하시오.

문제 12.7. 아래의 병렬프로그람을 작성하기 위하여 X3H5표기를 리용하시오.

(1) 계산문제
(2) 야끄비완화문제
(3) 가우스소거문제
(4) 목표탐색문제
이 장에서 명백하게 주지 않은 구체적인 표기들을 정의하시오.

문제 12.8. 리용의 간편성과 코드의 효과성의 견지에서 X3H5와 OpenMP를 비교하시오.

(1) X3H5로 그림 12−5를 다시 그리고 그림 12−5의 코드와 그것을 비교하시오.
(2) 계산을 위한 X3H5코드와 OpenMP코드를 (그림 12−6)비교하시오.

문제 12.9. 다음과 같은 문제들을 위한 병렬프로그람을 작성하기 위하여 SI Power C 표기술을 리용하시오

(1) 계산문제
(2) 야끄비완화문제
(3) 가우스서고문제
(4) 목표탐색문제
(5) 이 장에서 명백히 주지 않는 구체적인 표기를 정의하시오.

문제 12.10. 문제 12.6, 문제 12.7, 문제 12.9들에서 리용된 표기법들을 비교하되 사용의 간편성, 그것들이 병렬과 집중통신을 얼마나 잘 지원하는가 하는 견지에서 비교하시오.

문제 12.11. 왜 동기화구성이 필요한가를 보여 주는 확실한 실례를 드시오. 크로스바를 리용하여 밝히는데서 실례 12.11의 파라다임을 따르시오.

제１３장. 통보문넘기기프로그람작성

이 장에서는 처리되는 마디들사이에서 넘기기되는 통보문에 의한 병렬프로그람작성원리들을 상세하게 론의한다. 우리는 PVM과 MPI와 같은 두가지 널리 알려 진 령역통보문넘기기체계를 리용함으로써 병렬처리를 달성하기 위하여 어떻게 하여야 하는가를 론의한다. 이 두가지 인정되여 채용되고 있는 체계들은 여러 분야에서 얻은 연구결과이다. 이것들은 둘 다 선행한 프로그람작성체계들의 유용한 많은 특징들을 결합한것이다.

１３．１．통보문넘기기방식

이 절에서 분산형병렬성을 개발하기 위한 통보문넘기기수법을 특징 짓는다. 기초개념들은 12.2.2에서 이미 밝혔다. 보다 상세한것들은 이 절에서 설명한다.

１３．１．１．통보문넘기기서고

많은 통보문넘기기쏘프트웨어가 최근 몇해동안에 생성되였다. 가장 영향이 큰 마디들의 일부를 표 13-1에 제시하였다.

이 체계의 지적자는 참고문헌과 Web자원목록에서 찾아 볼수 있다.

표 13-1 통보문넘기기쏘프트웨어

이름	판매자	구별되는 특징
Cmmd	Thinking Machines	낮은 지연시간을 위한 활성통보문리용
Express	Parasoft	집중통신과 I/O
Fortran-M	Arginne National Lab	모듈성과 결정성
MPI	MPI Forum	널리 리용된 표준
Nx	Intel	Intel하이퍼립방체 Mpps로부터 시작
P4	Argonne National Lab	통합공유기억기와 통보문넘기기
PARMACS	ANL/GMD	주로 유럽에서 리용
PVM	Oak Ridge National Lab	광범히 리용, 단독체계
UNIFY	Mississippi State	MPI와 PVL호출을 허용하는 체계
Zipcode	Livermore National Lab	문맥개념에 기여함

초기 통보문넘기기체계에서 리용한 일부 좋은 착상은 PVM와 MPI표준들에 보충되였다.

말단사용자는 새로운 통보문넘기기응용프로그람을 개발하기 위하여 PVM 혹은 MPI들을 리용할것을 요구하였다. 기타 체계들은 널리 리용할수 없든가 이식할수 없는것으로 하여 대다수는 시대적으로 뒤떨어 진것으로 된다.

사실상 대다수 병렬콤퓨터판매자들은 PVM이든지 MPI를 선행한 독립적쏘프트웨어를 교체하여 통보문넘기기를 위한 고유한 지원체계로 리용한다.

이 묶음들에 대하여 아래에서 간단히 개괄한다.

독점적쏘프트웨어

CMMD[618]는 사유기계 CM−5체계에서 리용된 통보문넘기기형서고이다. 특징적인 사용자공간통신은 통신지연을 감소시키기 위한 능동통보문기구에 기초한다.

Parasoft회사에서 나온 Express software [372]은 프로그람작성환경이다.

이것은 병렬I/O 과 마찬가지로 점대점과 집체적통보문통신을 둘 다 지원한다.

Nx [502]는 Int의 MPP(즉 초립방형과 파라곤)를 위하여 개발된 극소핵심부체계이다.

이것은 Intel/Sandia ASCI TFLOP체계에서 PUMA라는 새로운 극소핵심부로 교체되였다.

공개령역 쏘프트웨어

Fortran−M[247]은 공유기억과 통보문넘기기를 둘 다 지원하기 위하여 설계된 Fotran 77에로 확장된것이다. 통보문넘기기지원만이 현재 실행되고 있다.

언어는 확정적인 동작을 제시하는 모듈명령프로그람개발을 촉진시키기 위한 많은 기구를 보장한다.

P4는 병렬처리기를 위한 병렬프로그람 [118]에 의거한다.

P4는 많은 구성방식들사이에서의 이식가능성을 의미한다.

PARMACS[30]는 P4로부터 개발된 통보문넘기기형묶음이다.

PVM과 MPI

대다수의 중요하고 대중적인 통보문넘기기쏘프트웨어묶음들은 MPI와 PVM들이다.

MPI는 MPI Forum에 의하여 개발된 함수서고의 표준적정의이다(MPI Forum은 병렬콤퓨터판매자들, 서고서술자들, 응용프로그람전문가들의 광범한 련합체이다.).

이 정의는 주요병렬콤퓨터판매자들전체가 리용한것이다.

PVM은 서로 다른 Unix콤퓨터망우에서 병렬응용프로그람실행을 위한 자체포함공개령역 쏘프트웨어체계이다.

시간이 지남에 따라 점점 일반화되므로 SMP들, PVP들, MPP들, 워크스테이션의 클라스터, 그리고 PC들에 적합하다.

13. 1. 2. 통보문넘기기모형

통보문넘기기체계에서는 모든 호상작용조작들을 통신, 동기화, 집중화라고 간주하면서 통신이라는 용어를 사용하는것이 습관으로 되여 있다.

따라서 2.4.2에서는 호상작용방식을 흔히 **통신화방식**이라고 부른다.

통신은 보통 같은 그룹사이에서 나타난다. 그러나 그룹사이의 통신은 또한 일부체계들(즉 MPI)에 의하여 지원된다.

이것은 사용자가 리해하여야 할 통신방식의 세가지 측면이다. 즉

- 얼마나 많은 프로쎄스들이 포함되는가?
- 프로쎄스들이 어떻게 동기화되는가?
- 통신완충기가 어떻게 관리되는가?

세가지 통신방식들은 현재의 통보문넘기기체계에서 리용된다.

우리는 리용자의 관점에서 아래에 세가지 서로 다른 방향의 송신(send)과 수신(receive)쌍을 리용한 이 통신방식들을 서술하기로 한다.

우리는 착상을 실현해 보기 위하여 다음과 같은 코드실례를 리용한다.

실례 13.1. 통보문넘기기에서 송신과 수신완충기

다음과 같은 코드로 프로쎄스 P가 변수 M에 포함된 통보문을 프로쎄스 Q에 송신한다. 여기서 Q는 변수 S에 포함된 통보문을 받는다.

```
프로쎄스 P:                     프로쎄스 Q:
M=10                            S=−100
L1:  send M to Q;               L1:  receive S form P;
L2;  M=20;                      L2:  X=S+1;
     Goto L1;
```

변수 M을 흔히 송신통보문완충기(혹은 **송신완충기**)라고 부르며 S를 수신통보문완충기(혹은 **수신완충기**)라고 부른다.

동기적통보문넘기기

프로쎄스 P가 동기적으로 *send* M **to** Q를 실행할 때 이것은 프로쎄스 Q가 동기적으로 *receive* S **from** P를 실행할 때까지 대기한다.

두 프로쎄스는 통보문 M이 송신되거나 수신될 때까지 송신통보문과 수신통보문으로부터 귀환되지 않는다.

이것은 우의 코드에서 변수 X가 11로 평가되여야 한다는것을 의미한다.

우의 코드에서 send와 receve가 귀환될 때 련이어 실행되는 명령문 L2에서 M은 P에 의하여 즉시 덧 씌여 질수 있으며 S는 Q에 의하여 즉시 읽혀 질수 있다.

동기통보넘기기에서는 추가적인 완충기가 필요 없다는것을 강조해 둔다. 수신통보문완충기 S는 들어 오는 통보를 잡을수 있다.

송신/수신차단

송신차단은 통보가 송신될 때까지 귀환되지 않는다. 이것은 통보변수 M이 안전하게 재차 씌여 질수 있다는것을 의미한다. 송신이 귀환될 때 그에 대응하는 수신은 시작할 필요도, 끝낼 필요도 없다는것을 언급해 둔다.

우리가 알고 있는것은 다만 통보가 M에서 탈퇴했다는것이다. 이것을 수신했을수도 있다. 그러나 망체계의 어떤 곳에 송신하는 상태에서 림시완충되여 있을수도 있으며 수

신마디의 완충기에 이것이 도달하였을수도 있다.

수신차단은 그에 대응하는 송신을 기다리지 않고 프로쎄스가 귀환될수 없다.

송신/수신차단에 의하여 우의 코드에서 X는 11로 평가되여야 한다. 체계는 통보문넘기기차단방식을 위한 림시완충기를 제공하여야 한다는것을 강조해 준다.

송신/수신비차단

송신비차단은 그에 대응하는 수신을 기다리지 않고 프로쎄스가 그것에 도달하면 실행된다. 송신비차단은 체계에 통보문 M을 송신하게 되여 있다는것을 알려 준 즉시 귀환될수 있다. 통보문자료가 M에서 탈퇴할 필요는 없다. 그러나 M을 덧 쓰는것은 안전하지 못하다.

수신비차단은 그에 대응하는 송신을 기다리지 않고 프로쎄스가 그에 도달하면 실행된다. 그것은 체계에 수신할 통보문이 있다는것을 알려 준 즉시 귀환될수 있다.

통보문은 이미 도착하였을수도 있고 오는 도중일수도 있으며 송신되지 않았을수도 있다. 비차단방식으로는 이 두가지 프로쎄스의 관계속도에 따라 X가 11,12 혹은 −99로 평가될수 있다.

체계는 비차단통보문넘기기를 위하여 림시완충기를 제공할수도 있다.

세가지 방식의 비교

이 세 방식은 표 13−2에서 비교한다.

동기방식은 두가지 주요우점을 가진다.

- 첫째로, 동기의미들이 명백하고 리용하기 쉽다는것이다. 프로쎄스가 송신루틴으로부터 귀환하면 통보가 송신되고 수신되였다는것을 명백하게 알수 있다.
- 둘째로, 분리된 자료완충기는 사용자나 체계 설정할 필요가 없다. 통보문 M은 수신자프로쎄스주소공간의 *S*로 직접 복사될수 있다. 결함은 송신자가 수신자의 대답을 기다려야 한다는것이다.

실제적인 병렬체계에서는 동기방식정의의 변수들이 있다.

실례로 일부 체계들에서 동기송신은 대응하는 수신이 시동되였지만 끝나기전에 시동될수 있다.

동기방식은 다른 뜻으로도 해석할수 있다. 비동기라는 말은 차단, 비차단방식과 같은 동기 아닌 방식에서 쓸수 있다. 차단, 비차단방식은 거의 모든 통보문넘기기체계에서 리용되고 있다.

그것들은 둘 다 동기송신대기시간을 줄인다. 그러나 비동기송신에서는 충분한 림시완충공간이 있어야 한다.

왜냐하면 그에 대응하는 수신이 시동되지조차 않았을수 있으므로 수신한 통보문을 저장해둘 기억공간을 알수 없기때문이다. 비차단방식은 대기시간을 최소한 줄인다. 그러나 그것은 추가적인 문제점을 야기시킨다.

실례 13.1에서 코드가 비차단 송신/수신을 리용한다고 가정하자.

명령문 M=20은 통보문 M이 송신되기전에 실행될수 있다.

따라서 새로운 값 20은 값 10대신에 Q에 넘기기된다.

수신자측에서 명령문 X=*S*+1는 P에서 통보문(10이든지 20의 둘가운데서 하나)이 *S*에 도달하기전에 실행될수 있다.

낡은 값으로 X의 −100이 리용될수 있다. 이와 같이 불필요한 동작을 수정하려면 통보문넘기기체계가 프로쎄스로 하여금 작업을 계속하기에 앞서 안전하게 될 때까지 기다리도록 억제하는 기능인 확인, 대기상태를 보장하여야 한다. 이런 기능으로는 실례 13.1의 코드가 아래와 같이 지적할수 있다.

표 13-2 세가지 통신모형들의 비교

통신사건	동기	차단	비차단
출발조건을 보내기	송신과 수신	송신	송신
보내기복귀를 가리킨다	통보문수신	통보문송신	초기화된 통보문송신
의미론	명백	중간	오유
완충통보문	요구되지 않음	요구	요구
상태검사	요구되지 않음	요구되지 않음	요구
대기부가처리	가장 높다	중간	가장 낮다
통신과 계산에서 중복	아니	예	예

```
프로쎄스 P:                         프로쎄스 Q:
    M=10;                               S=−100
L:  send M to Q:                        receive S form P;
    Do some computation                 do some computation
    Which does not change M             which does not use S
    Wait for M to be sent               wait for S to be received
    M=20;                               X=S+1
```

완충기공간은 체계가 자동적으로 제공할수도 있고(실례 SP2에서) 사용자가 직접 배정할수도 있다(실례 MPI에서). 이 두가지 경우에 사용자는 코드를 개발할 때 이 문제에 관심을 돌려야 한다. 만일 완충기공간이 충분히 보장되지 못하면 체계는 실패하거나 병렬프로그람이 교착된다. 비차단방식을 사용하게 된 주요한 동기는 뛰여넘기(overlap)통신과 계산을 수행할 필요가 제기된데 있다. 이것은 통보문이 넘기기될 때 프로쎄스가 통보문넘기기를 간섭함이 없이 련속 계산을 실행할수 있도록 한다. 이와 같은 뛰여넘기는 분산통보문프로쎄스가 달린 기계들에서 매력적이다. 그러나 비차단은 추가기억공간, 완충기배정, 완충기로서 통보문복사 및 꺼내기 그리고 추가대기기능실행과 같은 불합리성을 초래한다. 이러한 부가처리는 콤퓨터에 의한 뛰여넘기는 통신기의 리익이 없다. 또한 이와 같은 뛰여넘기효력을 적용할수 있을만한 충분한 자료를 보장하지 못한다. 따라서 이와 관련한 앞으로의 연구가 필요하다.

１３．２．통보문넘기기형대면부(MPI)

MPI는 통보문넘기기형함수의 서고를 위한 표준적인 정의(규정)이다.

MPI는 *MPI Forum*에 의하여 개발되였다. MPI는 공동령역(Public−Domain), 가동환경과 무관계한 통보문넘기기형서고는 표준형을 제공함으로써 이식불가능성을 가지게 된다.

MPI는 이 서고를 언어와 무관계한 형식으로 서술하며 포트란 C와의 결합을 보장한다. 이 정의는 조작체계, 하드웨어, 임의의 판매자들에만 국한된 어떤 특징도 포함하지 않는다.

이러한 리유로 하여 MPI는 병렬계산분야에서 널리 쓰이고 있다. MPI는 Windows를 리용하는 IDMPC의 모든 주요Unix워크스테이션, 모든 주요병렬콤퓨터들에서 실행되고 있다. 이것은 통보문넘기기를 위한 MPI를 리용하여 표준 C나 포트란우에서 작성된 병렬프로그람이 수정없이 단일 PC, 워크스테이션, 워크스테이션망, MPP, 임의의 판매자, 임의의 조작체계에 의하여 실행될수 있다는것을 의미한다.

PI설계의 우점은 MPI가 4개의 직교개념에 기초한 유력한 기능을 제공한다는데 있다. 사람들은 이 개념들을 개별적으로 배울수 있으며 이 개념들의 결합이 잘 정의된 의미론적패턴을 따른다는것을 알고 있다. MPI는 PVM보다 훨씬 더 많은 200개이상의 함수들을 보장한다. 그러나 저자들의 경험에 의하면 직교설계가 PVM보다 MPI가 배우고 리용하는데 쉽고 편리하다는것을 보여 준다.

4가지 개념은 통보문자료형태, 통신기들, 통신조작, 가상형태이다. 첫 세 개념을 고찰하면서 그것들이 통보문넘기기에서 가장 기초적이며 가장 널리 쓰인다는것을 알수 있다.

MPI에서 병렬문제

MPI는 프로쎄스가 시작할 때 어떻게 개입되며 실행되도록 하는가와 관련하여 아무것도 알려 주는것이 없다. 정적인 프로쎄스들을 가정하자. 즉 모든 프로쎄스가 병렬프로그람이 시동될 때 생겨난다고 하자. 이것들은 프로그람전체가 끝날 때까지 존재한다. 여기에는 MPI_COOMN_WORLD가 확인하는 모든 프로쎄스들로 이루어 진 그런 기정(default)프로쎄스그룹이 있다. 프로그람이 실행되고 있을 때에는 어떤 프로쎄스도 생겨 날수 없으며 중단될수도 없다. 그림 13−1을 통하여 해당한 문제들을 설명하는데 MPI프로그람을 리용하자. 이것은 $\sum f_{00}(i)$를 계산하기 위한 SPMD프로그람이다. 루틴 MPI_Init는 임의의 다른 MPI프로그람이 접근되기 전에 MPI를 초기화하기 위하여 모든 프로쎄스를 접근하여야 한다.

그다음 MPI_Comm_site와 MPI_Comm_rank루틴을 접근하여 기정그룹과 매 프로쎄스의 위수를 찾는다.

프로쎄스들은 통보문을 송신하고 수신하기 위하여 MPI_Send와 MPI_Recv루틴을 접근하여 서로 통신한다. 최종적으로 MPI기능이 더 필요 없게 되면 모든 프로쎄스들은 MPI_Finalize를 접근하여 MPI환경을 중단시킨다. 이 6개의 루틴은 MPI에서 완전한 통보문넘기기형프로그람을 작성하기 위한 최소모임을 구성한다. 그림 13−1에서 프로그람이 “myprog.c”파일에 보관된다고 생각할수 있다. IBM SP2다중콤퓨터체계에서 이 프로그람을 병렬C콤파일러

mpcc   mpcc myprog.c−O myprog

를 리용하여 콤파일된다.

실행될수 있는 myprog는 명령

MPIRUN myprog−np *n*

에 접근하여 *n*마디우에 적재된다. 같은 myprog *n*마디에서 *n*프로쎄스로써 적재될것이다. 마디프로그람들은 코드가 적재될 때 보게 되는 호스트파일에서 규정되게 된다. 이 *n*프로쎄스들은 프로그람이 끝날 때까지 존재한다. 그것들은 기정프로쎄스그룹인 MPI_COMM_WORLD를 이룬다. 프로쎄스 $k(k=0,1,\cdots,n-1)$를 위하여 MPI_Comm_size루틴은 group_size에서 *n*을 귀환시키며 MPI _ Comm _ rank는 my_rank에서 *k*를 귀환시킨다. 마디개수 *n*(실례로 기정프로쎄스그룹의 크기)은 적재시간파라메터이다. 같은 프로그람이 단 *n*마디에서 실행될수 있다. 프로쎄스 0은 우와 같은 코드에서 모든 I/0작업을 실행(출현)시킨

```
#include "mpi.h"
int foo(i) int i ; { ... }
main(argc, argv)
int argc;
char * argv[] ;
{   int i, tmp, sum = 0, group_size, my_rank, N;
    MPI_Init (&argc, &argv) ;
    MPI_Comm_size (MPI_COMM_WORLD, &group_size) ;
    MPI_Comm_rank (MPI_COMM_WORLD, &my_rank) ;
    if ( my_rank == 0 ) {
        printf("Enter N: "); scanf("%d", &N);
        for (i=1; i<group_size; i++)
S1:         MPI_Send(&N, 1, MPI_INT, i, i, MPI_COMM_WORLD) ;
        for ( i = my_rank ; i<N ; i = i + group_size )
            sum = sum + foo(i) ;
        for (i=1; i<group_size; i++) {
S2:         MPI_Recv(&tmp, 1, MPI_INT, i, i, MPI_COMM_WORLD, &status) ;
            sum = sum + tmp ;
        }
        printf ("\n The result = %d", sum) ;
    }
    else {      /* if my_rank != 0 */
S3:     MPI_Recv(&N, 1, MPI_INT, 0, i, MPI_COMM_WORLD, &status) ;
        for ( i = my_rank ; i<N ; i = i + group_size )
            sum = sum + foo(i) ;
S4:     MPI_Send(&sum, 1, MPI_INT, 0, i, MPI_COMM_WORLD) ;
    }
    MPI_Finalize() ;
}
```

그림 13−1. MPI 통보문넘기기형프로그람

다. 그것은 우선 사용자로부터 N의 값을 읽은 다음 그 값을 순환고리 for에 있는 다른 모든 프로쎄스들에 송신한다. 다음에 자기의 작업분배몫 $sum = foo(0) + foo(n) + foo(2n) \cdots$ 을 계산하며 다음에 다른 프로쎄스로부터 $n-1$부분합을 수신하여 그것들을 총 계산결과에 축적시킨다. 매개 다른 $n-1$개 프로쎄스들에 대하여 다음작업이 실행된다. 즉 프로쎄스 $k(0<k<n)$ 는 프로쎄스 0이 송신한 N을 수신한다. 일반적인 부분합 $sum = foo(k) + foo(n+k) + foo(2n+k) \cdots$ 을 계산하도록 작업분배를 계산한다. 다음에 이 부분합을 프로쎄스 0에 송신한다. 프로쎄스 0에 있는 send S1은 프로쎄스 *k*에 있는 receire S3과 정합되고 S4는 S1과 정합된다.

MPI실행

MPI는 단독으로 리용되는 쏘프트웨어체계가 아니라 이미 있는 병렬프로그람작성환경에서 통보문넘기기통신으로 봉사한다. 이것은 프로쎄스관리와 I/0와 같은 필수적인 기능들을 관리한다. 실례로 MPI는 IBM SP2우의 PCE/MPL의 정점(top) OSF/Nx에 설치될수 있다. 이 적합한 환경외에도 여러가지 대중적령역 그리고 Intel Parason우의 MPI환경들이 있다. 실례로 이미 Edinbug종합대학에서 개발된 CHIMP실행과 shio초대형콤퓨터쎈터에서 개발한 LAM(Local Area Multicomputer)을 들수 있다. 이것은 류다른 유닉스클라스터를 위한 MPI프로그람작성환경이다.

MPICH

이식가능이라는것은 같은 MPICH묶음을 꺼내여 그것을 다른 곳에 설치할수 있다는 것을 의미한다. MPI CH는 또한 많은 병렬기계에서 좋은 성능을 가진다.

１３．２．１．MPI통보문

MPI 프로쎄스들은 2.2.7에서 론의한 단일스레드화된 프로쎄스이다. 이것들은 분산된 극소공간을 가진다. 따라서 어떤 프로쎄스가 다른 프로쎄스주소공간에서 변수에 직접 접근할수 없다.

프로쎄스들사이의 통신은 통보문넘기기로 실행된다. 부분루틴 MPI_Send와 MPI−Recv은 그림 13− 1에 제시되여 있는바 이것은 프로쎄스쌍들사이에서 통보문을 넘기기하는 점대점통신조작이다. 이 부분루틴들은

```
MPI_Send(&N,1,MPI_INTi,i,MPI_Comm_WORLD);
MPI_Recv(&n,1,MPI_INT,O,i,MPI_COMM_WORLD,& status);
```

실례 13.1에 있는 송신 및 수신조작 즉

Send M to Q ; 와 receive S from P;

들보다 훨씬 복잡하다. MPI에서 왜 이런 복잡성이 제기되는가? 이 질문에 대답하기 위하여 우선 통보문이란 무엇인가에 대한 대답이 필요하다.

통보문이란 무엇인가?

비유하여 말하면 통보문은 편지와 같은것이다. 우리는 통보문내용을(편지 그자체) 지정할 필요가 있다. 그리고 우리는 편지에서처럼 편지를 받아야 할 대상(봉투에 씌여 진)을 규정하여야 한다. 전자를 **통보문완충기**, 후자를 **통보문봉투**라고 부른다.

실례 13.2. MPI를 거치는 자료의 통신배렬

C표기법 double A[100]로 선언된 N개의 합성수들의 배렬을 고찰하자. 이제 프로쎄스 P가 프로쎄스 Q에 다음과 같이 송신하려 한다고 하자. 즉

(1) 흠잡을데 없이 완전한 배렬 A;

(2) 배렬 A의 첫 두개의 원소(즉 A[0]과 A[n]);

(3) 배렬 A의 짝수첨수원소(즉 A[0],A[2],A[u],…)

경우 (1)은 단순한 Send A to Q를 실행하는 프로쎄스 P로 규정할수 있다.

통보문완충기는 배렬이름 A에 의하여 식별되며 봉투는 바로 프로쎄스이름 Q이다.

상업통보문체계에서 송신

IBM SP2에서 상업통보넘기기체계(즉 Intel paragon에서 NX와 IBM SP2에서 MPL)

send(*address, length, destination, tag*)

통보문송신루틴은 4가지 인수를 포함하여 더 유연하다.

여기서 통보문완충기는 (*address, length*) 쌍에 의하여 지정되며 통보문봉투는 (*destination, tag*)에 의하여 지정된다. 주소(*address*)마당은 통보문이 머물러 있는 기억부분의 시작주소를 지정한다. 그리고 길이부분은 얼마만한 바이트를 보내야 하는가를 지정한다. 콤퓨터에서 배정확도(double−precision number)가 64bit 또는 8bit라고 간주하자. 그러면 경우 (2)는 send(A, 16, Q, tag)에 의하여 지정할수 있다. C와 포트란에서 배렬의 시작주소는 배렬의 이름에 의하여 확인된다. 통보문길이는 두개의 배정확도를 포함하므로 16byte로 된다. 이 안(cscheme)을 가지고서는 경우 (3)을 원만하게 다룰수 없다.

MPI에서 송신(Send in MPI)

그림 13−1에 있는 MPI 송신부분루틴은 그림 13−2에 다시 제시하였다. 이 세가지 강조된 속성통보문주소, 통보문계산, 통보문자료형들은 통보문완충기와 다른 4개의 인수들이 통보문봉투를 형성하도록 지정한다. 주소마당은 자료구조의 시작주소가 되여야 한다는것은 아니다. 이것은 응용프로그람주소공간에서 아무런 기억주소라도 된다. 실례로 A의 셋째, 넷째, 다섯째 요소들을 송신할 때 우리는 **send**(A+2, 24, Q, tag)를 사용한다.

왜 통보문은 자료행인가

쌍 (*address, length*)은 MPI에 앞서 대다수 통보문넘기기체계에서 통보문완충기를 명

기하군 한다. 그러면 왜 MPI가 추가적인 통보문자료형속성을 소개하는가? 거기에 두가지 리유가 있다. 그 하나는 이종계산을 지원하기 위한데 있으며 다른 하나는 련속이 아니고 동일하지 못한 기억부분으로부터의 통보문들을 허용할수 있도록 하기 위해서이다.

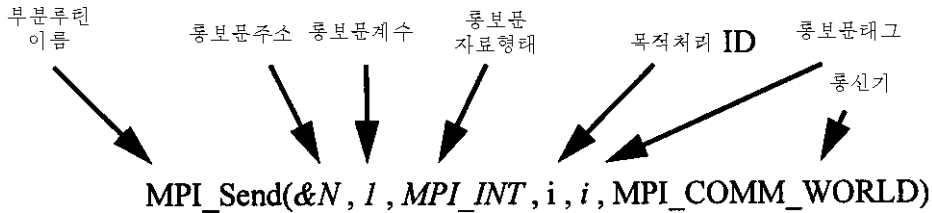



그림 13-2. 통보문송신에서 MPI성분들의 해부

류다른 계산이란 말은 워크스테이션망과 같은 그런 서로 다른 콤퓨터로 구성된 체계우에서 응용프로그람들의 실행과 관련되여 있다. 이 체계에서 매 콤퓨터들은 서로 다른 프로쎄스들과 조작체계를 리용하여 각이한 판매자로부터 제공될수 있다. 중요한것은 이 콤퓨터들이 서로 다른 자료표현방법을 사용하더라도 어떻게 호상실현가능성(interoperability)을 담보하는가 하는것이다. 어느 정도 극단한 경우를 고찰해 보자.

실례 13.3. 서로 다른 형의 자료송신

이제 워크스테이션 1에서 워크스테이션 2에 100개의 영문자들로 이루어 진 한개의 렬을 보내려 한다고 하자.

워크스테이션 1이 일정한 방법으로 **send**(String, 100, W2)를 실행할 때 워크스테이션 2는 그에 대응한 **recv**(String, 100, W1)을 실행한다.

여기서 W1은 워크스테이션 1, W2는 워크스테이션 2를 의미한다. 두 기계에서 자료구조렬은 **char** Stirng[1000]으로 규정 지어 진다. 그러나 이 방법은 실현할수 없다. 왜냐하면 W1과 W2의 **Char**자료형이 서로 다르게 정의되기때문이다. **Send**(String, 100, W2)의 의미는 시작주소렬을 리용하여 100byte(매 byte는 8bit)를 송신한다는것이다. 류사하게 **recv**(String, 100, W1)은 100byte를 수신한다는것을 의미한다. 이것은 W1기계에서도 좋다. 왜냐하면 문자가 바이트로도 표시되기때문이다. 그러나 W2의 기계에 있는 **Char**자료형은 W2에서 리용되는 모든 문자들(5만개이상)이 ASCII문자들과 호환되여야 한다. 따라서 W2에서의 매 문자는 16bit 또는 2개 byte자료항목이다. MPI는 표 13-3에 제시된것처럼 기초자료형을 미리 규정해 놓음으로써 이 문제점을 해결할수 있다. 이제는

W1 : MPI_Send(String, 100, MPI_CHAR, W2, …)
W2 : MPI_Recv(String, 100, MPI_CHAR, W1, …)

에 의하여 100개 문자를 통신할수 있다. 둘째 인수 100은 바이트로 볼 때 통보문길이를 넘지 않는다. 그러나 MPI_CHAR의 자료항목수는 길다. 이 량을 통보문길이와 구별하기 위하여 **통보문계산**이라고 부른다. 자료형MPI-CHAR는 W1의 콤퓨터우에서는 8bit로 실행할수 있으나 W2의 콤퓨터우에서는 16byte로 실행하여야 한다. 문자렬의 실행은 8bit

형식화(format)에서부터 16byte 형식화로 정확히 변환되도록 보장하여야 한다.

표 13-3　　　　　　　　MPI에서 자료형미리정하기

MPI (C 속박)	C	MPI (Fortran 속박)	Fortran
MPI_BYTE		MPI_BYTE	
MPI_CHAR	부호문자	MPI_CHARACTER	CHARACTER(1)
		MPI_COMPLEX	COMPLEX
MPI_DOUBLE	배정확도	MPI_DOUBLE_ PRECISION	DOUBLE_ PRECISION
MPI_FLOAT	단순정확도	MPI_REAL	REAL
MPI_INT	옹근수	MPI_INTEGER	INTEGER
		MPI_LOGICAL	LOGICAL
MPI_LONG	긴 옹근수		
MPI_LONG_DOUBLE	긴 배정확도		
MPI_PACKED		MPI_PACKED	
MPI_SHORT	짧은		
MPI_UNSIGNED _CHAR	비부호문자		
MPI_UNSIGNED	비부호옹근수		
MPI_UNSIGNED _LONG	긴 비부호		
MPI_UNSIGNED _SHORT	짧은 비부호		

MPI에서 미리 규정된 자료형들에는 C와 포트란의 모든 기초행과 두개의 추가행인 MPI_BYTE와 MPI_PACKED를 포함한다. MPI_BYTE는 8bit로 된 한개의 byte를 포함한다. MPI_PACKED들이 포함되였는가를 다음에 간단히 론의한다.

유도된 자료형

세가지 속성(주소, 계산, 자료형)을 가지도록 해도 실례 13.2에서 론의한 문제들을 풀기 어렵다. 즉 배렬 A의 모든 짝수첨수의 원소들을 송신한다.

실례 13.4. 비련쇄자료항목송신

비련쇄적인 자료항목을 송신하는 한가지 방도는 먼저 이 항목들을 련쇄적인 림시완충기에 묶어 놓고 그다음 그것을 전송하는것이다. 다음의 코드는 MPI를 리용하여 배렬 A의 짝수첨수요소들을 어떻게 묶어서 송신하는가를 설명한다.

```
Double  A[100];
MPI_Pack_size(50, MPI_DOUBLE, comm, &BufferSize);
TempBuffer=malloc(BufferSize);
j=sizeoff(MPI_DOUBLE);
Position=0;
For(i=0; i<50; i++)
   MPI_Pack(A+i*j, 1, MPI_COUBLE, TempBuffer, BufferSize, &Position, comm);
MPI_Send(TempBuffer, Position, MPI_PACKED, destination, tag, comm);
```

MPI_Pack_size루틴은 50개의 MPI_DOUBLE자료항목을 보존하는데 얼마나 큰 림시완충기가 요구되는가를 결정하는데 리용된다. 완충기크기는 TempBuffer에 기억기를 동적으로 배치하는데 리용된다. 배렬 A의 50개의 짝수첨수의 원소들이 **for**순환고리의 림시완충기에 정렬된다.

MPI_Pack루틴에서 첫 인수는 묶여 져야 할 배렬원소의 주소이다. 두번째 인수는 그 자료형이다. 변수position은 얼마나 많은 항목들의 자리길이 정렬되였는가를 보존하기 위해서 MPIpack루틴이 제정될 때 리용된다.

최종값은 통보문 총계로써 다음의 MPI_send에서 리용된다. MPI에서 지정된 모든 통보문들은 자료형MPI_PACKED를 리용한다는것을 강조해 둔다.

이와 같은 저장방법은 tedious와 error−prone이다. MPI는 사용자로 하여금 다음의 유연한 형식화로 통보문을 지정할수 있도록 한다. 통보문은 이런 여러개의 자료항목들로 구성되여 있으며 매개는 주소와 자료형(물론 자료값)을 가지고 있다.

실례 13.5. 통보문넘기기에서 혼합된 자료형송신

아래의 통보문에서 매개의 배정확도는 8byte이며 한 문자는 1byte이고 옹근수는 4byte라고 가정하자.

(1) 배정확도의 모든 요소들은 배렬 A에 있는 100개의 요소들이다. 이 통보문은 100개의 항목으로 구성되여 있다. 매 항목은 매 항목(8byte)의 크기를 결정하는 **double**자료형이다. i렬 항목에서 (시작)주소는 $A+8(i-1)$ 이다.

(2) 배렬 A의 셋째, 넷째 요소들. 이 통보문은 두개의 항목 A[2]와 A[3]으로 이루어진다. 매 항목은 **double**자료형을 가진다. 첫 항목은 A+16에서 시작되며 둘째 항목은 A+24에서 시작된다.

(3) 배렬 A의 짝수첨수요소들. 이 통보문은 50개 항목 A[0], A[2], A[4], A[98]들로 이루어 진다. 매 항목은 **double**자료형을 가진다. i째 항목의 주소는 $A+16(i-1)$ 이다.

(4) 문자 C가 따르고 다음에 또 옹근수 K가 따르는 배렬 A의 세번째 요소. 이 통보문은 서로 다른 자료형을 가지는 세개의 항목으로 이루어 진다.

이것은 구조의 부분 즉

Struct ｛**double** A[100]; **char** b, c; **int** j, k; ｝S;

이라고 가정한다. 따라서 주소는 첫 항목 (A[2])에 대하여 S+16이며 두번째 항목(C)에 대하여 S+801 그리고 세번째 항목(K)에 대하여 S+806이다.

(1)과 (2)의 통보문들은 두가지 성질을 가진다. 그 하나는 자료항목이 보존되여 있으며 모든 자료항목들이 같은 자료형을 가진다는것이다. 다른 하나는 이런 통보문들이 세가지 즉 address, count, datatype(주소, 계산, 자료형)에 의하여 편리하게 지정된다는것이다.

실례로 (1)에서 통보문은 (A, 100, MPI, DOUBLE)에 의하여 지정되며 (2)의 경우는 (A+16, 2, MPI_DOUBLE)에 의하여 지정된다. 그러나 이러한 단순한 방법으로는 (3), (4)의 경우를 다룰수 없다. (3)의 경우에 자료항목은 련쇄적인 위치에 머물러 있지 않는다. 경우 (4)에서 자료항목은 비련쇄적이며 혼합된 자료형으로 된다.

MPI는 유도된 자료형으로 되였을수 있는 비련쇄적인 자료항목들로 구성된 임의의 통보문에 대하여 지정하도록 도출된 자료형의 개념을 도입하였다. 도출된 자료형은 사용자응용프로그람이 실행될 때 기본자료형으로 이루어 진다. MPI는 매우 유력하며 복잡한 자료행을 구성할수 있는 부분루틴(Versatile subroutine)들을 가진다. 기본취지를 해설하기 위하여 실례를 리용하기로 한다.

실례 13.6. 배렬의 모든 짝수첨수를 가지는 요소들의 송신

이 실례는 첨수화된 배렬의 부분을 어떻게 조종하겠는가를 보여 준다.

```
Double A[100];
MPI_Data_type EvenElements;
… …
MPI_Type_vector(50, 1, 2, MPI_DOUBLE, &EvenElements);
MPI_Type_commit(&EvenElements);
MPI_Send(A, 1, EvenElements, destination, …);
```

프로그람작성자는 새로운 자료형인 EvenElement를 선언하여 MPI_Data_type를 사용한다. MPI_Type_vector(*count*, *blocklength*, *stride*, *oldtype*, *&newtype*)는 도출된 자료형을 구성하기 위한 하나의 MPI루틴이다.

도출된 형인 *newtype*는 *block*의 *count copy*들로 이루어 진다.

매 블로크는 차례로 현존 자료형으로써의 낡은 형인 항목들의 *blocklength*부분들로 이루어 진다. *stride*는 매 두블로크들사이에 있는 *oldtype*요소들의 개수를 지정한다. 따라서 (*Stride*−*blocklength*)는 두 블로크사이에 있는 gap이다.

MPI_Type_Vector(50, 1, 2, MPI_DOUBLE, &EvenElements)루틴은 50개의 블로크로 이루어 진 도출된 자료형EvenElement를 만들어 낸다. 매 블로크는 한개의 배정확도로

구성되며 여기에 8byte 틈이 따르고 그뒤로 다음블로크가 따른다.

Stride는 두개의 배정확도 또는 16byte의 크기를 가진다. 이 새로운 행은 송신루틴에서 리용되기에 앞서 기억되여야 한다. EvenElement의 매 한개의 요소는 배렬 A의 짝수첨수요소 50을 모두 포함한다는것을 강조하게 된다. 따라서 계산부분은 MPI−Send에서 값 1을 가진다.

통보문완충기

통보문완충기(혹은 간단히 완충기)는 통보문넘기기분야에서 서로 다른 의미로 리용되여 왔다. 아래에서 몇가지 실례를 들기로 하자.

- 완충기는 통보문자료값이 보존되여 있는 프로그람작성자가 지적한 응용프로그람기억지대를 념두에 둔다. 실례로 **Send**(A, 16, Q, tag)의 완충기 A는 사용자응용프로그람에서 다르게 선언된다.
- 완충기는 또한 통보문이 송신될 때 림시로 보존하는 통보문넘기기체계에 의하여 만들어 지고 운영되는 일부 기억지대를 의미한다. 이런 완충기는 사용자응용프로그람에서 나타나지 않으며 때때로 체계통보문완충기(혹은 완충기체계, system buffer)로 불리워 진다.
- MPI는 세번째 가능성을 허용한다. 사용자는 중간완충기로 하여금 응용프로그람에서 나타날수 있는 통보문을 저장하기 위하여 일정한 범위의 기억지대를 설정할수 있다.

실례 13.7. 프로쎄스쌍사이의 통보문송신

프로쎄스 P의 배렬 A에 보존된 통보문 M을 프로쎄스 Q의 배렬 B에로 전송하기 위한 다음과 같은 코드를 고찰하자. 즉 코드는

프로쎄스 P : 프로쎄스 Q :

double A[2000000] **double** B[32]
send(A, 32, Q, tag) **recv**(B, 32, P, tag)

와 같다.

세가지 형의 완충기리용이 그림 13−3에 제시되였다. 여기서 M은 배렬 A의 첫 4개 요소들을 포함하는 32byte통보문을 지정한다.

1)의 경우에 통보문은 사용자준위의 완충기 A와 B들사이의 변환을 지령한다. 이때 완충기들은 둘 다 사용자응용프로그람에서 선언된다. 수신완충기는 충분히 커야 한다.

실례로 **send**(A, 64, Q, tag)가 P에 의하여 실행되는 경우에 오유가 발생할수 있다. 이러한 오유는 현재의 통보문넘기기체계로써는 발견할수 없으므로 오유수정에서 많은 애로가 제기된다.

비동기통보문넘기기로써는 우에서 지적한(ㄱ)경우) 직접적인 송신을 실행하기 힘들다.

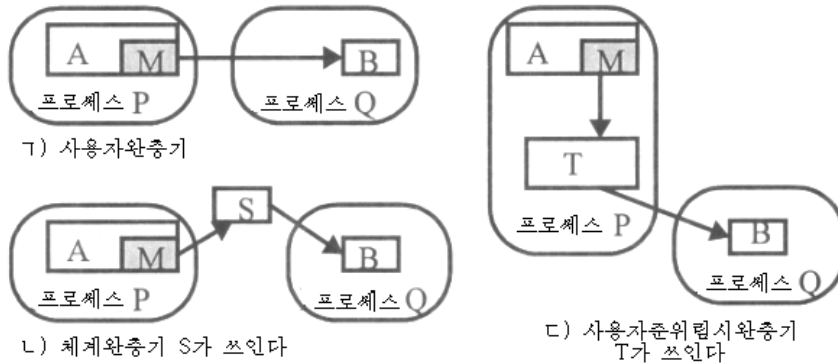



그림 13-3. MPI에서 사용되는 통보문완충기의 세가지 형

일련의 대책안들이 제기되였다.

2)의 경우에 통보문은 수신사용자완충기에 들어 가기 전에 동적으로 생성된 체계완충기 S에 림시로 복사된다. 이와 관련하여 두가지 문제점이 제기된다. 그것은 첫째로, 지나친 복사는 추가적인 부가처리를 초래하며 둘째로, 충분한 크기를 가진 체계완충기를 담보할수 없다는것이다. 필요한 완충기는 물리적기억장치를 지나치게 크게 요구하므로 파잉결과를 초래한다. 또한 실제적인 기억용량에 비하여 완충기가 지나치게 커서 프로그람이 중지되거나 중단된다.

(3)의 경우는 MPI에서 사용되는 또 다른 방도이다. 사용자는 우선 완충시켜야 할 임의의 통보문을 저장하는데 충분한 림시완충기 T를 설정한다. 통보문넘기기작업이 실행될 때 통보문은 우선 수신자완충기 B에 기입되기전에 완충기 T에 복사된다. 만일 체계의 완충기 T를 보상할수 없는 경우에 응용프로그람으로 하여금 오유통보문을 검사하게 하고 작업을 중단시킨다. 그렇지 않으면 응용프로그람은 통보문완충기공간을 담보하여야 한다.

13.2.2. MPI에서 통보문봉투

통보문을 어떻게 받아 들이려 하는가?

아래에 목록화된것은 통보문봉투가(Massage Envelope) 세개의 입력량들로 구성되는 MPI송신루틴이다. 목적지마당은 통보문이 송신되여야 할 처리를 의미하는 옹근수이다.

MPI_Send(address,count,datatype,*destination*,*tag*,*communicator*)

왜 꼬리표가 필요한가?

통보문형식으로 알려 진 통보문꼬리표는 사용자가 통보문의 각이한 형태들에 이름을 붙여 통보문접수를 속박시키는데 리용되는 옹근수이다. 왜 꼬리표가 필요한가를 보기 위해서 먼저 꼬리표없는 다음의 코드를 고찰하자.

프로쎄스 P:	프로쎄스 Q:
Send(A, 32, Q)	**recv**(X, P, 32)
Send(B, 16, Q)	**recv**(Y, P, 16)

이 코드의 목적은 A의 첫 32byte를 X에, B의 첫 16bit를 Y에 전송하는것이다. 통보문 B가 후에 송신되지만 프로쎄스 Q에는 더 일찌기 도착하므로 첫 **recv**()에 의하여 X에 수신된다. 이런 오유는 다음과 같은 꼬리표들을 리용하여 피할수 있다.

```
프로쎄스 P:                      프로쎄스 Q:
Send(A, 32, Q, tag1)             recv(X, P, tag1, 32)
Send(B, 16, Q, tag2)             recv(Y, P, tag2, 16)
```

새로운 통보문 A(꼬리표 1을 가진)는 먼저 수신되도록 담보된다. 통보문 B가 Q에 먼저 도착하면 그것은 **recv**(Y, P, tag2, 16)가 실행될 때까지 대기한다.

실례 13.8. 통보문넘기기에서 리용된 꼬리표들

꼬리표를 리용하는 또 다른 리유를 다음실례에서 볼수 있다. 봉사기프로쎄스 Q에 봉사요청통보문을 송신하는 요청프로쎄스 P와 R를 가정하자.

```
프로쎄스 P:                      프로쎄스 R:
Send(request1, 32, Q)            Send(request2, 32, Q)
```

어느 송신이 먼저 실행될것인가는 미정이다. 봉사기처리 Q는 의뢰기요청이 먼저 들어 와 먼저 봉사되는 순서로 처리할것을 요구한다.

```
프로쎄스 Q:
while(true){
    recv(received_request, Any_프로쎄스, 32) ;
    프로쎄스 received_request;
}
```

이 코드는 모든 요청에 같은 방법으로 처리되지 않으므로 그리 유연하지 못하다. 꼬리표를 리용하면 서로 다른 통보문들을 서로 다르게 처리할수 있다.

```
프로쎄스 P :                     프로쎄스 R :
Send(request1, 32,Q,tag1)        send(request2,32,Q,tag2)

프로쎄스 Q :
While(true){
    Recv(received_request,Any, 프로쎄스, 32, Any_tag, Status)
    If(Status.  Tag==tag1)프로쎄스 received_request in one way;
    If(Status.  Tag==tag2)프로쎄스 received_request in another way;
}
```

recv()명령문은 임의의 처리로부터(Any_Process는 *Wild_card_Process ID*로 알려 져 있

다.) received−request에로 임의의 꼬리표(Any−Tag는 Wild−card−tag로 알려 져 있다.)를 가진 32bit의 통보문을 수신할것이라는것을 알린다. 수신된 통보문의 실지 꼬리표는 프로그람에서 가지분기를 일으키는 Tag field of Status에 기억된다.

통신기란 무엇인가?

통신기란 하나의 프로쎄스 group와 Context이다. 하나의 프로쎄스그룹은 프로쎄스들의 정렬된 유한모임이다.

유한성은 그룹크기로 불리우는 *n*이 유한이라는 의미이다. 따라서 우리는 유한 *n*개의 프로쎄스들의 모임을 념두에 둔다. 순서 붙인다는것은 *n*개의 프로쎄스들을 옹근수 0, 1, ⋯, *n*−1들로 등급분류한다는것을 의미한다.

매 프로쎄스에는 통신기(그룹)안에서 자기 순위가 대응되여 있다. 그림 13−1에 제시한것처럼 그룹크기와 프로쎄스순위는 MPI루틴을 접근함으로써 얻을수 있다.

즉

```
MPI_Comm_Size(Communicator,&group_size)
MPI_Comm_rack(Communicator,& my_rank)
```

거의 대다수 MPI사용자들은 한개의 그룹(MPI에서 *intra−communicators*라고 부른다.) 안에서 통신을 위한 루틴만을 요구하여야 한다. 이것은 모든 MPI통신루틴은 통신기인수를 가진다는것을 의미한다. MPI는 내부련결통신기(*inter−communicator*)를 통한 내부련결그룹(inter−group)통신을 허용한다. 이에 대하여서는 더 론의하지 않고 독자들에게 [575]와 MPI홈페지를 지적한다. MPI에서 *Context*들은 호상 복잡한 통신속에서 서로 다른 통신내용을 안전하게 분류하는 체계형supertag와 같다. 매 통신기는 특유한 문맥을 가진다. 한개의 문맥으로 송신된 통보문은 다른 문맥으로는 수신할수 없다. 문맥개념이 왜 필요한가를 보기 위하여 다음의 실례를 고찰하자

실례 13.9. 통신기리용

다음과 같은 코드일부를 보자

Consider the following code fragment;

```
Process 0:
MPI_Send(msg1, count1, MPI_INT, 1, tag1, comm1);
Parallel_fft(⋯);

Process 1:
MPI_Recv(msg1, count1, MPI_INT, 0, tag1, comm1);
Parallel_fft(⋯);
```

목적은 프로쎄스 0이 프로쎄스 1에 msg 1을 송신하며 둘 다 부분루틴 parallel_fft()을

실행하는데 있다. 이제 parallel_fft()이 다른 송신부분루틴 즉

if(my_rank==0)MPI_SendCmsg2, count 1, MPI_INT, 1, tag 2, comm 2

를 포함한다고 가정하자.

통신기가 없었다면 프로쎄스 1에서 MPI−Recv는 tag 2가 tag1과 같은 옹근수값을 가질 때 parallel_fft()에로 MPI_Send가 송신한 msg 2를 잘못 수신할수 있다. 그러면 왜 우리가 다른 tag값을 사용할수 없는가?

이와 관련하여 세가지 리유로 tag1과 tag2의 명백성을 담보하기 어렵다. 그것은

- tag들은 사용자가 지정한 옹근수값이며 사용자들자체의 실수도 있다.
- 사용자의 실수가 없다 하여도 tag1의 값이 tag2의 값과 다르다는것을 담보하기가 어렵거나 불가능하다. Parallel_fft()기능은 다른 사용자가 쓸수도 있으며 서고루틴이 될수도 있다. 따라서 사용자는 tag2의 값을 알지 못할수 있다.
- 사용자가 tag2의 값을 안다 하여도 오유가 계속 나타날수 있다.
  따라서 Mpi_Recv부분루틴은 임의의 무질서하게 카드화된 tag MPI−Any−tag를 리용하기로 결심하기때문이다.

MPI는 이 모든 문제들을 통신기를 리용하여 해결한다.

Parallel_fft에서 통신은 같은 프로쎄스그룹(즉 프로쎄스 0과 1)에 들어 있으면서도 매 통신기가 고유하며 체계에 지정된 문맥(context)이 Comm1의것과 서로 차이나는 통신기들을 사용한다. 따라서 MPI_Eecv가 부주의로 Mpi_Send가 송신한 msg2를 우연히 수신할 위험성은 없다. MPI는 서로 다른 통신기를 분리시켜 통신을 보장하도록 설계되였으므로 임의의 집중통신은 그것들이 같은 종류의 통신기라 할지라도 점대점통신으로 분리시킨다. 이 통신기개념은 다른것과 함께 병렬서고개발을 도모한다.

통신기

MPI는 미리 정의된 여러개의 통신기들을 가지고 있다. 실제로 MPI_COMM_WORLD는 모든 프로쎄스들의 모임을 포함하며 부분루틴 MPI_Init가 실행된 다음에 정의된다. MPI_COMM_SELF는 그것을 사용하는 과정만을 포함한다. MPI는 또한 사용자가 정의한 통신기들을 설치하기 위한 여러개의 부분루틴들을 제공한다. 우리는 실험을 통하여 그중 두가지만을 설명한다.

실례 13.10. MPI에서 새로운 통신기

10개의 프로쎄스가 실행되는 다음의 코드를 고찰하자.

```
MPI_Comm My World, SplitWorld;
Int          my_rank, group_size, Color, Key;
MPI_Init(&argc,&argv);
```

```
MPI_Comm_dup(MPI_COMM_WORLD,&My World);
MPI_Comm_rank(MyWorld, &my_rank);
MPI_Comm_size(MyWorld, &group_size);
Color=my_rank%3;
Key=my_rank/3;
MPI_Comm_split(MyWorld, Color,Key,&SplitWorld);
```

루틴 MPI_Comm_dup(MPI COMM_WORLD, My World)실행은 원본 MPI_COMM_WORLD와 같이 10개의 프로세스들로 되여 있지만 서로 다른 문맥을 가진 같은 그룹을 포함하고 있는 새로운 통신기 My World를 생성한다.

MPI_COMM_split가 어떻게 작업하는가를 표 13- 4에 제시하였다.

My World통신기의 10개 프로세스들은 3개의 통신기로 분할된다.

같은 색을 가진 모든 프로세스들로 Split world라는 새로운 통신기를 구성하였다. 그것들은 새로운 통신기에서 건반에 의하여 주어 진 순서대로 순위를 정하였다. 색에 서로 다른 3개의 값이 나타나는것으로 하여 3개의 통신기가 형성된다. 모두가 다 Split World라고 이름을 달았는데 그것들은 서로 다른 통신기들이다.

표 13-4　　분할통신기 My world

Myworld에서 렬	0	1	2	3	4	5	6	7	8	9
색갈	0	1	2	0	1	2	0	1	2	0
열쇠	0	0	0	1	1	1	2	2	2	3
분할 world에서 렬 (색=0)	0			1			2			3
분할 world에서 렬 (색=1)		0			1			2		
분할 world에서 렬 (색=2)			0			1			2	

통보문특징요약

일반적인 형태 즉

```
MPI_Send(buffer, count, datatype, destination,tag,communicator)
```

로 송신부분루틴의 통보문특징들을 요약하자.

그리고 그림 10-1과 그림 13-2에 있는 명백한 실례로써

```
MPI_Send(&N, 1, MPI_INT, i, i, MPI_COMM_WORLD)
```

를 고찰하자.

이것은 표시 1이 꼬리표 i를 가지고 장소 &N에 기억된 옹근수이며 프로세스는 그것을 송신한다는것을 의미한다.

송신기와 목적프로세스들은 둘 다 모든 프로세스들이 통신기에 존재한다. 이것은 아

래와 같이 개괄할수 있다.

- 첫 인수는 통보문완충기의 첫 주소를 가리킨다.
- 둘째 인수는 일정한 자료형마다 많은 항목들이 통보문에 포함되는가를 규정한다.
- 자료형은 기본형이나 파생형일수 있다.
- 넷째 인수는 목적프로쎄스 ID(렬)이며 다섯째는 통보꼬리표이다.
- 여섯째 항목은 프로쎄스그룹과 문맥, 통신기를 동일시한다. 보통 통보문들은 같은 그룹의 처리기들사이에서만 넘기기된다. 그러나 MPI는 통신기들사이를 통하여 그룹호상간의 정보통신을 허용한다.

대응하는 수신부분루틴은 송신부분루틴과 매우 류사하다:

```
MPI_Recv(&tmp,1,MPI_INT,i,i,MPI_COMM_WORLD,&Status)
```

혹은 더 일반적으로

```
MPI_Recv(address, count, datatype, source, tag, communicator, status)
```

첫 인수는 수신통보문완충기시작주소 즉 수신하여야 할 자료보존용기억지대를 지정한다. 둘째 인수는 수신가능한 셋째 인수의 주어 진 일정한 자료형의 최대항목수를 지정한다. 수신해야 할 실제적인 항목수는 그보다 작을수 있다. 넷째 인수는 원천프로쎄스 ID(rank)이며 다섯째 인수는 통보문 tag이다. 이 두 부분은 무질서하게 카드화된 MPI_Any_source와 MPI_Any_tag일수 있다. 여섯째 인수는 통신기를 확인한다. 일곱째 인수는 수신된 통보문에 대한 여러가지 정보를 보존하고 있는 구조를 가리키는 지적자이다.

```
MPI_Status Status
```

이것은 수신된 통보문에 대한 여러가지 정보를 가진다. 실례로 실지 원천프로쎄스렬과 실지 통보문꼬리표는 두 마당 MPI_SOURCE와 MPI_TAG에서 찾아 볼수 있다. 수신된 자료항목의 실제개수는 다른 MPI부분루틴을 통하여 알아 볼수 있다.

```
MPI_Get_count(&Status, MPI_INT, &C);
```

부분루틴은 Status의 정보를 리용하며 일정한 자료형(이 경우에 MPI_INT)의 항목의 실제개수를 결정하여 이 수자를 변수 C에 배치한다.

실례 13.11. 통보문넘기기에서 상태

상태정보는 서로 다른 프로쎄스들로부터 서로 다른 크기와 tag를 가진 통보문을 수신하는데 유익하다. 다시 한번 실례 13.8의 의뢰기-봉사기프로그람을 보자.

일련의 봉사기프로쎄스들은 봉사기에 통보문을 송신하여 봉사를 요구한다. 거기에는 통보문 tag가 확인하는 세가지 형의 봉사기가 있다. 입력파라메터는 통보문의 실제적인 자료항목에 해당하는 요구와 같다. 따라서 서로 다른 통보문들은 서로 다른 크기를 가진다. 같은 봉사기를 요구하는 두개의 통보문은 같은 tag를 리용한다.

```
While(true){
    MPI_Recv(received_request, 100,MPI_BYTE, MPI_Any_source, PI_Any_tag, comm. ,
&Status);
    Switch(Status. MPI_Tag){
        Case  tag_0:  perform service type 0;
        Case  tag_1:  perform service type 1;
        Case  tag_2:  perform service type 2;
    }
}
```

13. 2. 3. 점대점통신

MPI는 차단 및 비차단송신/수신 두 조작을 다 보장한다. 즉 하나의 차단송신(수신)은 통보문완충기가 안전하게 쓰기(읽기)할 때까지 귀환될수 없다. 그 반면에 비차단송신(수신)은 즉시 귀환될수 있다.

MPI는 차단/비차단속성의 서로 다른 결합과 몇가지 서로 다른 통보문완충기사용방식에 기초하여 많은 량의 송신(수신)부분루틴을 보장한다.

통신모형들

통신모형이라는 말은 송수신사이의 완충기운영과 하나의 동기화방법을 의미한다. 이 말의 뜻은 10. 1. 2의것과 다르다. 이 모형들은 차단과 비차단에서 송신조작에 적용된다. MPI는 일반적인 차단 및 비차단수신을 가지는바 이것은 13.1.2에서 론의하였다.

- **동기화** 송신은 그에 대응하는 수신이 시작될 때까지 귀환될수 없으므로 응용프로그람완충기는 도착하는 통보문을 보존하기 위하여 수신자측에 제공하여야 한다. MPI에서 비차단동기송신은 불가능하다. 그것은 귀환되였다 하여 통보문이 송신되였다는것을 의미하지 않기때문이다. 수신자측에서 추가적으로 완충기를 설치할 필요는 없다. 체계완충기는 송신자측에서 여전히 요구할수 있다. 통보문이 지나치게 복사되는것을 제거하기 위하여 차단동기송신을 리용하여야 한다.
- **완충화** 완충화된 송신은 일정한 량의 완충공간의 리용가능성을 가정할수 있다. 이것은 바이트크기의 사용자완충기를 배당하는 부분루틴호출 MPI_Buffer_

attach(buffer, size)를 통하여 사용자프로그람에 의해 미리 규정되여야 한다. 이 완충기는 MPI _ Buffer _ detach(*buffer*,*size*)에 의하여 해제된다. 이 완충기방식은 그림 13-3 ㄴ)에 제시된다.

- **표준화** 실행에 따라 동기 혹은 완충방식으로 송신할수 있다.
- **준비** 수신측이 이미 응답하였으면 송신을 계속 진행한다. 이런 송신은 동기방식에서는 대기하지 말아야 한다. 이것은 일부 경우에 보다 효과적인 통신규약을 사용하기 위하여 리용한다.

이 방식들은 블로크화속성과 함께 표 13-5에서 보는바와 같이 MPI에서 여덟개의 부분루틴들을 발생한다. 이때 수신루틴은 두개뿐이다.

표 13-5 MPI에서 각이한 송신/수신조작들

MPI	차단	비차단
표준송신	MPI_Send	MPI_Isend
동기송신	MPI_Ssend	MPI_Issend
완충송신	MPI_Bsend	MPI_Ibsend
준비송신	MPI_Rsend	MPI_Irsend
수신	MPI_Recv	MPI_Irecv
검사완료	MPI_Wait	MPI_Test

비차단통신

MPI에는 4개의 비차단송신루틴들과 한개의 수신루틴이 있다. 그것들을 초기(시동)송신이나 수신에 리용한다. MPI는 송신이나 수신의 완료를 검사하는 다른 루틴들을 제공한다. 실례로 두가지 기본루틴들을 론의하자.

실례 13.12. 통보문넘기기를 리용하는 프로쎄스관흐름

그림 13-14에 제시된것은 왼쪽으로부터 입력자료를 련속적으로 수신하고 새로운 자료렬을 계산하며 그것을 오른쪽으로 송신하는 세 프로쎄스관흐름이다.

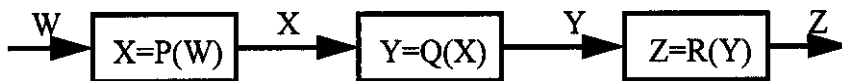



그림 13-4. 프로쎄스관흐름에서 자료렬의 흐름

```
while(Not_Done){
   MPI_Irecv(NextX, …) ;
   MPI_Isend(PreviousY, …) ;
   CurrentY=Q(CurrentX) ; }
```

프로쎄스 Q는 프로쎄스 P로부터 다음의 자료항목 X를 수신하는 비차단수신을 수행

하며 비차단은 선행자료항목 Y를 프로쎄스 R에로 송신하는것을 초기화하기 위하여 송신한다. 이때 현재의 X의 계산은 송신/수신이 진행되는 동안에 수행된다. 몇가지 주의사항이 있다. 우선 X와 Y를 위한 두개의 분리된 완충기들이 필요하며 다음 X와 Y의 수신에 대하여 완충기에 머무르는 동안 계산은 다른 완충기에 있는 현재 X를 사용할수 있다. 이러한 2중완충조직도해는 잘 알려 진 기술로서 그림 13- 5에 제시되였다.

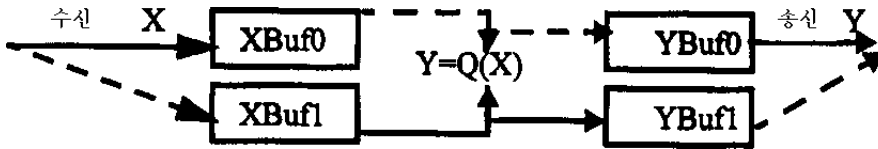



그림 13-5. 프로쎄스판흐름에서의 2중완충화

고정된 흐름선들은 순환고리의 반복에서 완충기들이 어떻게 리용되는가를 보여 준다. 이것들은 다음반복에서의 실선들로 절환된다. 다음으로 완충기의 자료는 그것이 갱신되기전에 리용된다.

```
While(Not_Done){
   If(X==XBuf0){X=XBuf1;Y=YBuf1;Xin=XBuf();Yout=YBuf0;}
   Else{X=XBuf0;Y=YBuf0;Xin=XBuf1;Yout=YBuf1;}
   MPI_Irecv(Xin,⋮,recv_handle);
   MPI_Isend(Yout,⋮,send_handle);
   Y=Q(X);/**/
   MPI_Wait(recv_handle,recv_status);
   MPI_Wait(send_handle,send_status);
   }
```

if-else명령문은 2중완충기를 절환하는데 리용된다. 루틴을 송신하는 비차단코드는 송신완료를 검사하는데 리용되는 보충적인 파라메터송신조종을 제외하고 블로크화전송과 같은 형태를 가진다.

MPI_Isend(buf,count,datatype,dest,tag,comm. ,send_handle)

마찬가지로 비차단수신루틴 MPI_Irecv는 한개이상의 파라메터수신조종을 가진다. 송신/수신의 완료검사는 루틴코드를 통하여 실행되는바 이 루틴은 조종에 의하여 지정되는 송신이나 수신이 완료되기전까지 귀환하지 않으며 일부 상태정보는 파라메터상태로 넘겨진다.

MPI_Wait(Handle,Status)

그러므로 코드는 두개의 비차단송신과 수신이 완료되기전까지 다음반복에로 이행하

지 않는다. MPI는 또한 조종에 의하여 지정되는 송신과 수신이 완료되였는가를 검열하는 다른 루틴코드를 제공한다.

MPI_Test(Handle,Flag,Status)

그러면 기발에 True값을 배정한다.

실례 13.13. 송신과 수신통보문들에서의 교착
다음의 코드가 실행될 때 어떤 현상이 일어 나는가?
그 코드는 몇가지 오유들을 가지는데 목적지향성 있게 작성된다.

프로쎄스 P:	프로쎄스 Q:
… …	… …
X=0;	if(Y==5)MPI_Irecv(&Y,⋯,P,⋯);
MPI_Issend(&X,⋯,Q,⋯);	printf(“Y is %d”,Y);

이것은 다음과 같은 결과들이 존재할 때에는 오유코드로 된다.

- 교착, 루틴MPI−Issend는 비차단 및 동기송신이다. 이 루틴은 응답(대응)MPI−Irecv가 실행되기 전에는 귀환되지 않는다. 이런 정황은 조건 Y=5이 진실이 아닐 때 생긴다.
- Y가 0인 경우 if명령문이 실행되면 Y=5이다. 이때 MPI−Irecv는 프로쎄스 P로부터 X값(0)을 자체의 Y에로 수신할수 있으며 값을 출력한다.
- Y가 5인 경우 Y=5일 가능한 경우(씨나리오)는 X값이 Y에 수신되기전에 printf명령문이 수행되는 경우이다(MPI−Irecv는 비차단 수신이라는것을 주의하시오.). Y의 낡은 값이 출력된다.

13. 2. 4. 집체적 MPI 통신

집중통신은 전역통신조작과 관련되는 모든 프로쎄스그룹을 포괄한다. 1HPI는 표 10−6에 제시한바와 같이 풍부한 집중통신루틴 2들을 제공한다.
하나의 프로쎄스그룹 Comm이 *n*개의 프로쎄스를 포함한다고 가정하고 몇가지 루틴들을 론의함으로써 기본적인 집중통신원리들을 자세히 보기로 하자.

방송
다음의 방송조작에서 루트(Root)를 련결시키는 프로쎄스는 콤프로쎄스(Comm process) 그룹에서 자기자체를 포함한 모든 프로쎄스들에 같은 통보문을 통신한다.

MPI_Bcast(Address,Count,Datatype,Root,Comm)

표 13-6 MPI에서 집중통신

형 태	루 틴	기 능 성
자료이동	**MPI_Bcast**	하나 대 전체, 식별통보문
	MPI_Gather	전체 대 하나, 개별통보문
	MPI_Gatherv	MPI_Gather, 일반화
	MPI_Allgather	MPI_Gather, 일반화
	MPI_Allgatherv	MPI_Allgather, 일반화
	MPI_Scatter	하나 대 전체, 개별통보문
	MPI_Scatterv	MPI_Scatter, 일반화
	MPI_Alltoall	전체 대 전체, 개별통보문
	MPI_Alltoallv	MPI_Alltoall, 일반화
집 합	**MPI_Reduce**	전체 대 하나 감소
	MPI_Allreduce	MPI_Reduce, 일반화
	MPI_Reduce_scatter	MPI_Reduce, 일반화
	MPI_Scan	전체 대 전체 병렬접두사
동기화	**MPI_Barrier**	장벽동기화

통보문의 내용은 점대점통신에서와 마찬가지로 세개의 변수(주소, 계산, 자료형)에 의하여 규정된다.

루트프로쎄스에 대하여 이 세개의 변수들은 송신완충기와 수신완충기를 규정한다. 다른 프로쎄스들에 대하여 이 세개의 변수는 수신완충기를 규정한다.

모으기

모으기루틴은 다음과 같이 주어 진다.

MPI_Gather(SendAddress, SendCount, SendDatatype, RecvAddress, RecvCount, RecvDatatype, Root, Comm)

Root 프로쎄스는 자체를 포함하여 모든 *n*개의 프로쎄스로부터 전용화된 통보문을 접수한다.

이 *n*개의 통보문들은 등급순서로 련결되며 root프로쎄스의 수신완충기에 보존된다.

매 송신완충기는 송신구조, 송신계산, 송신자료형과 같은 3원무이로 식별된다. 수신완충기는 모든 non-Root프로쎄스들에 대하여 무시된다. Root프로쎄스들에 대하여 이것은 3원무이(수신주소, 수신계산, 수신자료형)로 식별된다.

산란(Scatter)

*Scatter*루틴(routine)은 아래와 같다.

MPI_Scatter(SendAddress,SendCount,SendDatatype, RecvAddress, RecvCount, RecvDatatype, Root, Comm)

MPi−Scatter(송신주소, 송신계산, 송신자료형, 수신주소, 수신계산, 수신자료형, Root, Comm)과 같이 주어 진다.

산란은 모으기와 상반되는 조작을 수행한다. Root프로쎄스는 자기자체를 포함하여 n개 프로쎄스들에 개별적통보문을 송신한다. 이 n개의 통보문들은 원래 Root프로쎄스의 송신완충기에 순위에 따라 보존된다. 매 수신완충기들은 3원무이(RecvAddress, RecvCount, RecvDatatype)(수신주소, 수신계산, 수신자료형)에 의하여 식별된다. 송신완충기는 모든 non−Root프로쎄스에서는 무시된다. Root프로쎄스에 대하여 이것은 3원무이(SendAddress, SendCount, SendDatatype)(송신주소, 송신계산, 송신자료형)에 의하여 확인된다.

전체 교환(Total Exchange)

전체 대 전체의 총 교환루틴 즉

MPI_Alltoall(SendAddress, SendCount, SendDatatype, RecvAddress, RecvCount, RecvDatatype, Comm)

에서 모든 프로쎄스는 자기자체를 포함하여 n개의 프로쎄스들가운데서 매개에 개별적통보문을 송신한다. 이 통보문들은 송신완충기에 원래의 순위에 따라 보존된다. 다른 측면에서 통신을 고찰해 보면 모든 프로쎄스들이 개개가 n개의 프로쎄스들로부터 통보문을 수신한다는것을 알수 있다. 이 n개의 통보문들은 순위에 따라 련결되여 수신완충기에 보존된다. 총 교환은 n개의 모으기와 같으며 그것은 서로 다른 프로쎄스에 의하여 진행된다. 따라서 Root인수는 더는 필요 없게 된다. 전체 대 전체에서 n^2개의 통보문이 총 교환으로 통신된다.

집합화(Aggregation)

MPI는 두가지 형의 집합화 즉 축소와 주사를 제공한다(2.4.1 볼것).

MPI축소루틴의 문법은 다음과 같다.

MPI_Reduce(sendAddress, RecvAddress, Count, Datatype, Op, root, Comm)

여기서 매 프로쎄스들은 SendAddress에 보관된 값을 유지한다. 모든 프로쎄스들은 이 값들을 최종결과로 감소시키며 Root 프로쎄스의 RecevAddress에 보존한다. 자료항목의 자료형은 Datatype마당으로 규정한다. 축소조작자는 축소와 류사한 문법으로 되여 있다.

MPI_Scan(SendAddress, RecvAddress, Count, Datatype, Op, Comm)

Root마당은 없으며 따라서 주사는 n개의 최종결과들에 대한 값들을 결합하여 n개 프로쎄스 RecvAddress에 보관한다. MPI축소 및 주사루틴들은 매 프로쎄스들의 벡토르를 기증할수 있다. 벡토르의 길이는 Count로 규정된다.

MPI는 리용자가 정의하는 축소 혹은 주사조작을 지원한다.

장벽

장벽조작 MPI−Barrier(Comm)에서 통신기 Comm의 모든 프로쎄스들은 서로 동기화되여 있다. 즉 프로쎄스들은 개별적인 MPi−Barrier와 함수가 실행되기전까지는 대기한다.

집체적루틴의 일반적특징들

MPI에서 모든 집중통신조작은 다음과 같은 몇가지 공통적인 특징을 가진다.

- 통신기의 모든 프로쎄스들은 집중통신루틴을 반드시 접근해야 한다. 일부 통신기성원들이 집체적루틴을 호출하는 코드는 다른것들이 그것을 접근하지 않는 동안은 오유이다. 오유코드의 동작은 명백치 않으며 교착 혹은 틀린 결과를 포함하여 임의의것을 내보낼수 있다.
- MPI Barrier을 위한 모든 집체적루틴은 점대점통신에서 표준과 류사한 통신방식과 차단방식을 리용한다. 즉 집중조작의 참여가 끝나자마자 프로쎄스는 집체적루틴으로부터 귀환된다. 실례로 Root프로쎄스가 MPI_Bcast에서 귀환될 때 보내기완충주소는 안전하게 다시 리용될수 있다는것을 의미한다. 다른 프로쎄스들은 그것의 대응하는 MPI_Bcast로 아직 출발조차 할수 없다.
- 집체적루틴은 그 실행에 따라 동기화하든지 동기화하지 않을수 있다. MPI는 사용자가 동기화실행이 선택되든 선택되지 않든 코드가 정확하다는것을 담보하는 응답성을 취할것을 요구한다.
- 계산과 자료형은 관련되는 모든 프로쎄스우에서 호환（정합）되여야 한다.
- 집체적루틴에는 tag인수가 없다. 통보문봉투는 통신기인수와 원천/목적지프로쎄스들을 지정한다. 실례로 MPI_Bcast에서 통보문의 원천은 Root이며 목적지는 프로쎄스(Root를 포함하여)전체이다.

실례 13.14. MPI 프로그람에서 배렬조종

세개의 프로쎄스로 실행되는 MPI를 고찰하자. 이것들은 모두 초기에 다음의 코드를 실행한다.

```
Int i, j, my_rank, group, size, A[3], B[3], tag=1, root=0;
MPI_Comm comm;
MPI_Init(&argc,&argv);
comm=MPI_COMM_WORLD;
MPI_Comm_rank(comm. ,&my_rank);
MPI_Momm_size(comm. ,&group_size);
for(i=0;i<3;i++){A[i]=B[i]=my_rank*group_size+i;}
```

다음의 코드부분의 매개가 실행된 이후에 프로쎄스 2에서 A[1], B[0]의 값들이 얼마인가?

```
(1) if(my_rank==0){
        MPI_Bcast(A, 3, MPI_INT, root, comm);
        MPI_Send(B, 3, MPI_INT, 2, tag, comm);
}else if(my_rank==1){
        MPI_Bcast(A, 3, MPI_INT, root, comm) ;
Else{
        MPI_Recv(B, 3, MPI_INT, 0, tag, comm) ;
        MPI_Bcast(A, 3, MPI_INT, root, comm) ;
}
```

```
(2) MPI_Bcast(A,3,MPI_INT,root,comm);
(3) MPI_Scan(A,B,1,MPI_INT,MPI_SUM,comm);
(4) MPI_Scatter(A,1,MPI_INT,B,1,MPI_INT,root,comm);
```

풀이 초기코드가 실행된후 배렬 A와 B는 다음의 값을 가진다.

표 13-7 3개의 프로세스에서 배렬값

	프로쎄스 0	프로쎄스 1	프로쎄스 2
A[1]=B[1]	0	3	6
A[1]=B[1]	1	4	7
A[1]=B[1]	2	5	8

(1) 코드는 프로쎄스 0의 배렬 B를 프로쎄스 2로 보내고 프로쎄스 0의 배렬 A를 모든 프로쎄스들로 방송하려고 한다. 따라서 답은 프로쎄스 2에서 A[1]=1, B[0]=0이다. 그러나 이 코드는 오유이다. 왜냐하면 MPI_Bcast가 동기적으로 실행될 때 교착되기때문이다. 따라서 이 코드는 MPI-정합이 아니며 결과는 명백치 않다.

(2) 프로쎄스 2에서 A[1]=1, B[0]=6

(3) 이 조작은 A[0]의 （0, 3, 6）을 주사하고 프로쎄스 0에서 B[0]=0, 프로쎄스 1에서 B[0]=0+3=3, 프로쎄스 2에서 B[0]0=0+3+6=9를 생성한다.

(1)의 루틴은 프로쎄스 0에서 배렬 A의 3개 원소를 모든 3개 프로쎄스로 산란시킨다(프로쎄스당 한개 원소가 대응되도록). 대답은 프로쎄스 2에서 A[1]=7과 B[0]=2이다.

１３．２．５．MPI-2 확장

MPI는 1994년에 규격화되면서 거의 모든 병렬체계에서 리용되였다. 실행가동환경의 범위는 MPP콤퓨터와 SMP봉사기로부터 워크스테이션과 PC의 클라스터까지이다. 그러나 사용자는 더 많은 기능이 추가될것을 요구한다. 1997년에 MPI연산은 MPI-2라는 수정된 규격을 발표하였다. 원래의 MPI의 이름을 MPI1로 개칭하였다. MPI2는 많은 새로운 특징들을 추가하였으며 사용자/판매자공동체는 그 모든 특징들을 포섭하기 어려울수 있다. 이 문제를 취급하기 위하여 동적프로쎄스와 일방적인 통신을 리용한다. 따라서 간단히 다른 MPI-2의 특징들을 개괄한다.

동적프로쎄스

MPI-2의 병렬프로그람에서 프로쎄스들은 정적이다. 즉 프로그람은 주어 진 개수의 프로쎄스들을 가지고 시동하며 프로그람의 실행도중에 그 어떤 프로쎄스도 추가하거나 제거할수 없다. MPI-2은 동적프로쎄스를 지원한다.

그 우점은 다음과 같다.

- MPI-1은 프로쎄스들이 어떻게 생성되며 그것들이 통신을 어떻게 형성하는가를 밝히지 못한다.그러므로 MPI-1은 SP2에서의 POE, 워크스테이션망에서와 같은 능력을 보장하기 위한 기초적인 가동환경을 요구한다. MPI-2에서 동적프로쎄스 트랜잭션은 이식가능한（가동환경에 무관계한） 방식으로 이 능력을 보장한다.
- 동적프로쎄스는 MPI에 PVM코드를 이식하는것을 촉진한다. 그것들은 의뢰기-봉사기와 프로쎄스조와 같은 중요한 응용클라스터들을 리용가능하게 한다.
- 동적프로쎄스는 자원과 부하균형을 더 효과적으로 리용하게 한다. 례를 들어 리용된 마디점은 필요에 따라 줄거나 늘어 난다.
- 고장허용도 지원될수 있다. 한 마디점이 실패하면 그우에서 실행하는 프로쎄스는 다른 마디점에 만들어 진 새로운 프로쎄스에로 장애넘기할수 있다.

MPI-2동적프로쎄스운영에 일련의 새로운 기능을 추가해 준다. 다음의 형태를 가진 MPI_Spawn에 대하여 론의한다.

```
Int  MPI_Spawn(
      Char* command_line,          /*  실행가능한것과 인수*/
      Int  minprocs,               /*  실행해야 할 과제의 최소수*/
      Int  maxprocs,               /*  실행해야 할 과제의 최대수*/
      Char*info,                   /*  프로쎄스출발방법과 장소*/
      Int  root,                   /*  뿌리(부모)프로쎄스의 위수*/
      MPI_Comm  comm,              /*  뿌리과제의 통신자*/
      MPI_Comm*intercomm,          /*  통신자와 새로 생성된 프로쎄스사이의 호상
                                       통신자*/
      Int*array_of_errcodes        /*  생성된 과제당 오유통보*/
      )
```

이 기능은 *maxprocs*라는 자식프로쎄스를 퍼뜨리려고 노력한다. 매개는 *command_line* 에 지적된 꼭 같은 코드를 실행한다. 만일 MPI가 *maxprocs*프로쎄스를 퍼뜨릴수 없으면 *minprocs*프로쎄스를 퍼뜨리게 할수 있다. MPI_Spawn가 *minprocs*프로쎄스를 만들지 못하면 오유가 발생된다. *Info*인수가 어디서 언제 프로쎄스를 시동시켜야 하는가를 지정할수 있도록 실행시간체계에 맡긴다.

*Array_of_errcodes*는 길이 *maxprocs*의 배렬이다. 12개의 프로쎄스가 성과적으로 시동될 때 *array_of_crrcodes*의 첫 *k* 개 요소들 매개에 MPI_Success가 포함되게 되는 한편 나머지 매 요소들은 프로쎄스가 왜 성과적으로 시동되지 못하는가 하는 원인을 알려 주는 오유코드를 포함하게 된다.

MPI_Spawn은 집체적인 조작으로서 이것은 통신기 *comm*의 모든 성원들이 그 프로쎄스를 생성하도록 접근되여야 한다는것을 의미한다.

그러나 Root프로쎄스에서 인수들인 *command_line*, *minprocs*, *maxprocs*와 *info*들만이 의의가 있으며 다른 프로쎄스우에서 이 인수값은 무시된다. 따라서 *comm*에서 퍼뜨리기프로쎄스는 모든 프로쎄스에 대하여 자식으로 간주할수 있지만 뿌리프로쎄스만이 진짜 부모이다. 자식들을 MPI_Spawn으로 집중되는 MPI_Initialize로 호출하여야 한다는것을 의미한다. 자식과 부모프로쎄스들은 intercomm이라는 호상통신기를 통하여 통신한다.

이것들은 부모프로쎄스에 있는 MPI_Spawn에 의하여 귀환되며 자식프로쎄스는 MPI_Parent기능을 접근함으로써 호상 통신기를 조종할수 있다. MPI _ parent는

```
int MPI_parent(MPI_comm. *intercomm)
```

형태를 가진다.

일방적통신

MPI−2에는 일방적통신기능을 실행할수 있도록 하는 프로쎄스인 원격기억접근(RMA)으로 불리우는 새로운 점대점통신모형이 포함되여 있다. 여기서는 한 프로쎄스가 다른 측의 참가없이 원천으로부터 자료값을 꺼낼수 있으며 목적지에 송신할수도 있다. 이것은 MPI−1과 대조된다. MPI−1에서 모든 점대점통신은 쌍방적(two−sided)이다.

이때에는 송신자와 수신자프로쎄스들이 둘 다 참가하여야 한다. MPI−2는 RMA를 지원하기 위하여 풍부한 기능을 포함하고 있다. 세가지 RMA기능만 보자.

원격조작전에 기억기창문을 설정하여야 한다.

이것은 MPI−RMA기능으로 실행되는데 다음과 같다.

```
Int  MPI_RMA_init(
     Void*window_base,         /*  기억기창문의 기초주소*/
     Int  window_size,         /*  바이트단위의 기억기창문크기*/
     Int  disp_unit,           /*  변위에 해당한 단위의 크기*/
     MPI_Comm  comm,           /*  현재의 통신자*/
     MPI_Comm*newcomm          /*  RMA용 새 통신자*/
     )
```

통신기 *comm*에서 모든 프로쎄스들은 같은 MPI_RMA_init를 집체적으로 찾아야 한다. 이 집체적호출은 *newcomm*이라는 새로운 통신기에 귀환된다. 매 접근프로쎄스는 *newcomm*에서 임의의 프로쎄스에 의하여 원격접근할수 있다는것을 자기의 주소공간의 창문에 지정하여야 한다. 창문은 *window_base*에서 시동되며 *window_size*_byte만한 크기를 가진다. 임의의 RMA조작은 *disp_unit*라는 하나의 단위로 창문에 접근되여야 한다.

실례로 *disp_unit*=8일 때 RMA는 *window_base*+8, *window_base*+16 등에서 시동되는 자료완충기에 접근할수 있다. 그러나 *window_base*+10에서 되지 않는다.

일방적복귀는 *get*로 불리운다. MPI_Get는 다음의 형태를 가진다.

```
int MPI_Get(void*dest_addr,MPI_Init dest_count,
   MPI_Datatype dest_datatype,int source_rank,int source_disp,
   int source_count,MPI_Datatype source_datatype,
   MPI_Comm comm,
 )
```

이 함수는 원천처리기억기창문의 자료완충기의 내용을 목적지프로쎄스(호출프로쎄스)의 자료완충기에 전송한다.

원천프로쎄스는 comm과 *source_rank*쌍에 의하여 일의적으로 지정된다.

원천기억기창문은 이미 있는 MPI_RMA_init호출에서 실제적으로 *newcomm*인 *comm*에 의하여 결정된다.

원천자료완충기는 *window_base*+*source_disp***disp_ unit*에서 시동된다. 이것은 *source_datatype*형의 자료항목 *soruce_count*를 포함한다. 목적지자료완충기는 *dest_addr*에서 시동되며 *dest_datatype*형의 자료항목 *dest_count*을 포함한다.

MPI_Get는 차단기능이다. 이것은 자료가 목적지자료완충기에 도착한 다음에 귀환된다.

일방적송신은 put에 의하여 호출된다.

MPI_put는 MPI_Get와 매우 류사하다. 즉

```
int MPI_Get(void*source_addr,MPI_Init source_count,
     MPI_Datatype source_datatype, int dest_rank,int dest_disp,
     int dest_count, MPI_Datatype dest_datatype,
     MPI_Comm comm,
 )
```

이다.

차이점은 자료전송방향이 반대라는것뿐이다.

자료는 목적지기억기창문을 완충한 원천자료의 복사이다.

MPI_Get는 차단기능이다. 이것은 자료가 원천자료완충기에서 떠난후에 귀환된다.

기타 MPI확장자들

동적프로쎄스와 일방적통신외에 MPI-2는 MPI-1을 다른 측면에서 확장해 준다. 아래에서 그 부분의 명세를 라렬한다.

- 집중통신은 비차단방식과 호상통신기를 포함시키기 위하여 확장하였다. MPI_1에서는 다만 차단과 호상통신기에서 집체적인 통신이 지원되였다.
- MPI_2는 확대가능한 I/O가 보충되였다. 그것을 MPI-IO라고 부른다. MPI-1에서 I/O문제는 무시된다.
- MPI-1에서는 다만 포트란 99와 C를 위해서만 국한되여 있다. MPI-2에서는 포트란 90과 C++이상으로 확장되였다.
- MPI-2는 실시간프로쎄스를 위한 지원이 보충되였다.
- MPI-2는 MPI-1의 외부대면부를 확장하여 MPI대상에 대한 보다 접근가능한 환경도구쓰기장치를 주었다.
- MPI-2는 MPI-1를 확장하여 비차단집중통신호상통신기들을 허용하였다.

１３．３．병렬가상기계(PVM)

PVM은 원래 Unix콤퓨터만이 대규모의 통보문넘기기병렬콤퓨터로 리용되도록 설계된 자체포함형공개령역 쏘프트웨어체계이다. 인기가 높아 지면서 그것은 SMD, PVP, MPP 그리고 워크스테이션과 PC의 클라스터에 이식되였다. PVM은 Windows NT와 Windows 95와 같은 비Unix가동환경에 대하여 실현되였다. 프로그람지원언어는 C, 포트란과 Java이다.

개발력사

PVM의 개발은 1989년에 Oak Ridge국립연구소에서 시작되였다. 그 연구 및 개발은 아직도 종합대학들과 연구소의 연구사들을 망라하는 계획이다. 비록 더욱더 많은 사람들이 MPI를 리용하고 있다 하지만 PVM은 여전히 가장 일반적인 병렬처리용쏘프트웨어가동환경이다.

PVM에 의하여 사용자는 완전히 련결마디점들의 모임 즉 가상기계를 구성할수 있다. 매 마디점은 순차식, 벡토르 혹은 병렬콤퓨터와 같은 임의의 Unix콤퓨터가 될수 있다. 그래서 사용자는 많은 프로쎄스들을 동적으로 만들고 관리하며 이 가상기계에서 실행할수 있다. PVM은 프로쎄스사이의 통신과 다른 기능을 지원하는 서고루틴을 제공한다.

MPI와의 비교

PVM과 MPI의 기본차이는 PVM이 독립적인 체계라고 할 때 MPI(보다 명백하게는 MPI-1)는 그렇지 못하다는것이다. MPI는 프로쎄스관리와 I/O기능을 보장하기 위하여 중요한 장치에 의존한다. 이 기능은 PVM에 포함되여 있다. 반면에 MPI는 통보문넘기기에서 보다 유력하다. 그러나 MPI와 PVM은 표준이 아니다. PVM은 이식가능하지 못하다. 지난 2년간 PVM은 안정되여 있다.

이제부터 PVM에 대하여 고찰하자.

１３．３．１．가상기계구성

PVM체계는 두개 부분으로 이루어 진다. 즉 가상기계의 매 콤퓨터에 상주하는 PVM데몬(pvmd라고 부른다.)과 프로쎄스관리, 통보문넘기기 그리고 가상기계관리와 련결되는 사용자접근가능한 서고（libpvm3a라고 부른다.）로 이루어 진다.

기본사용을 보여 주는 몇개의 단순한 실례를 리용한다. PVM인 이 설계들이 보여 주는 것보다 훨씬 강력하고 유연하다. PVM을 어떻게 설치하고 리용하는가는 PVM사용자안내를 보면 된다. PVM에서는 마디를 **호스트**라고 부른다. 또한 프로쎄스를 **과제**라고 부른다.

PVM조종탁

PVM을 설치한후 사용자는 호스트로부터 PVM조종탁을 창조하는 다음과 같은 명령을 결정한다.

```
Pvm  host_file
```

이 명령을 성과적으로 실행하자면 호출호스트와 임의의 host_file에서 렬거된 매 호스트에서 출발해야 하며 호출호스트(마스터 host라고 부른다.)에서 다음과 같은 prompt를 현시해야 한다.

```
Pvm>
```

이것은 호스트가 PVM조종탁방식에 있다는것을 가리킨다. PVM조종탁은 자립형이고 호상작용PVM프로쎄스이며 shell과 류사하다. 사용자는 가상기계를 관리하고 PVM응용일감을 호출하며 일감실행을 감시하는 명령들을 전형으로 한다.

표 13.8 에 자주 쓰이는 PVM 조종탁명령들을 렬거한다.

동적구성

가상기계는 PVM서고함수들을 호출하는 사용자응용프로그람에 의하여 동적으로 구성될수 있다. Pvm_addhosts와 pvm_delhosts함수들을 하나 혹은 그이상의 호스트(host)들을 가상기계에 더하거나 더는데 쓴다.

```
int info, nhost=2, infos[2];
Char *hosts[]={“apple”, “orange.Usc. edu”}
info=pvm_addhosts(hosts, nhost, infos);
info=pvm_delhosts(hosts, nhosts, infos);
```

이 함수들은 두개의 호스트 “apple”과 “orange”를 각각 첨가하거나 제한한다. 호스트이름은 생략할수도 있다.

표 13-8 주요 PVM조종탁명령

명령	동작의미
pvm> add apple	호스트 "apple"을 가상기계에 첨가
pvm> delete apple	호스트 "apple"을 가상기계로부터 삭제
pvm> conf	가상기계구성을 렬거
pvm> spawn -count 4 app	가상기계에서 "app"을 실행시키기 위한 4개과제를 출발
pvm> jobs	가상기계에서 동작하는 일감렬거
pvm>halt	모든 PVM를 중지하고 PVM을 끌낸다

변수 *hosts*는 호스트의 이름이고 *nhost*첨가되였거나 제거한 호스트수이다.옹근수렬 *infos*의 길이 *nhost*는 매 호스트에 되돌아 오는 상태코드를 말한다.

부수값은 오유조건을 가리킨다. 이 두개의 함수들이 첨가되였거나 제거된 호스트수들을 되돌려 보낸다. 완전히 성공하면 info는 nhost와 같아 진다.

기구와 실행

출발된 첫번째 pvmd를 마스터pvmd라고 부른다. 다른 pvmd데몬들은 마스터(master)에 의해 련속 출발되는데 슬라브(slave)라고 부른다. 마스터와 슬라브데몬들의 모임은 가상기계를 이룬다. 가상기계는 모든 시간에 정확히 한개 마스터를 가진다.

최소가상기계는 바로 하나의 성원 즉 마스터로 이루어 진다. 단지 마스터는 가상기계로부터 슬라브를 첨가하거나 제거할수 있다. 그러나 첨가 혹은 제거슬라브에 대한 요청은 마스터데몬외부에서 온다. 그러한 요청은 언제나 마스터데몬을 넘기기해서 특정한 호스트의 슬라브를 출발시킨다. PVM은 마스터가 **rsh**, **rexec**(), 그리고 다른것들을 거쳐서 슬라브를 출발시키게 한다.

호스트표라고 부르는 자료구조는 매 호스트에 거주한다. 호스트표는 가상기계의 매 호스트에 대한 호스트서술자라고 부르는 입력을 가진다. 호스트서술자는 통신을 위한 파케트대기렬과 통보문완충기는 물론 호스트구성정보를 가지고 있다. 초기에 호스트표는 마스터host에 대한 단 한개의 입력자료를 가진다.

새로운 슬라브가 가상기계에 첨가될 때 마스터host의 호스트표는 새롭게 첨가된 슬라브를 위한 새로운 입력을 포함하도록 갱신된다. 그다음 갱신된 호스트표의 정보는 전체 가상기계에 발송되여 새롭게 첨가된 슬라브host들을 포함한다. 이렇게 하여 가상기계의 모든 호스트의 호스트표들은 호스트충돌과 망고장을 제외하고 동기화되고 일치된다.

가상기계가 어떻게 재구성되는가를 보기 위하여 호스트가 그림 13-6에서 어떻게 첨가되는가를 토론하자.

가상기계가 두개의 호스트 H1과 H2로 이루어 졌다고 가정하자.

호스트 H2에서 PVM과제는 새로운 슬라브host H3을 가상기계에 첨가하기 위하여 PVM서고함수 pvm_addhosts()를 접근한다. 이 통보문은 호스트 H2의 PVM데몬 pvmd2을 넘기기하여 함수 dm_addhost()를 접근한 다음 마스터pvmd1을 넘기기한다.

마스터데몬은 pvmd3을 출발하기 위해 **rsh**(혹은 다른 수단)를 리용하며 슬라브구성정

보를 후에 얻는다. 그다음 마스터데몬은 구성정보를 분석하고 호스트데몬을 갱신한다. 결과들은 호스트표를 갱신하기 위해 슬라브에 방송한다. 완성하기 위하여 응답을 마스터로 보낸다.

그림 13-6. 동적호스트 H3첨가

마스터가 모든 호스트표들이 갱신되였다는것을 결정한후 통보문을 방송하여 모든 슬라브들이 제각기 호스트표들을 가지게끔 한다. 동시에 마스터는 요청슬라브에 add_host동작이 완성되였다는것을 통보한다. 과제 T는 pvm_addhost가 완료된 국부데몬에서 통보를 받을 때까지 차단한다(대기한다.).

13. 3. 2. PVM에서 프로쎄스관리

PVM은 정적 그리고 동적병렬성을 가능하게 한다. 정적병렬프로그람을 출발시키기 위하여 사용자는 PVM조종탁에서 "spawn_count 4 *foo*"를 실행할수 있다.

이것은 4개 과제 (프로쎄스) 의 병렬응용을 창조하여 가상기계에서 실행한다. 매 과제는 같은 코드 *foo*를 실행한다. 응용은 *foo*가 동적과제창조함수 pvm_spawn()을 접근하지 않으면 정적DOP 4를 가진다.

명령 "spawn_count 4 *foo* "는 4개 과제 SPMD프로그람이 실행되는데 필요하다는것만을 보여 준다. 과제들이 어떻게 호스트에 넘겨 지는가 즉 어느 과제가 어느 호스트에서 실행되는가를 보여 주지 않는다.

PVM은 과제를 호스트로 넘기는 배당알고리듬을 제공한다. PVM은 사용자가 매 과제에 대한 특정호스트 혹은 구성방식을 명백히 규정하도록 한다. 실례로 조종탁명령

pvm>spawn_(*apple*)*foo*

은 host *apple*에서 과제를 시작하여 코드 *foo*를 실행한다. 명령

pvm>spawn_(*RS6K*)*foo*

은 AIX조작체계를 리용하는 RS/6000마디에서 과제를 시작한다.

표 13-9 프로쎄스관리를 위한 PVM함수

PVM 함수호출	의미
tid=pvm_mytid();	호출과제의 ID얻기
tid=pvm_parent();	parent과제의 ID얻기
info=pvm_exit();	호출과제 PVM에 존재 Unix처리로서 과제는 련속 동작한다.
numt =pvm_spawn(...);	PVM과제발생
info=pvm_kill(tid);	PVM과제끝내기
tstat=pvm_pstat(tid);	PVM과제상태얻기
info=pvm_tasks(...);	가상기계에서 동작하는 모든 과제정보얻기
mstat=pvm_mstat(host);	호스트상태얻기
info=pvm_config(...);	전체가상기계의 구성정보얻기

프로쎄스관리함수

PVM은 MPMD병렬성과 동적프로쎄스관리를 허용한다. 가장 중요한 함수는 pvm_spawn()이다.

이것과 다른 프로쎄스관리함수일부를 표 13-8에 렬거한다.

pvm_spawn함수를 자세히 보자. C에서 함수견본과 개요는 다음과 같다.

Int *numt* // 출발과제의 실제개수

=PVM_spawn(

Char **progm*, // 실행가능한 파일이름

Char ***argv*, // 인수배렬지적자

Int *flag*, //어느 호스트에 생성하겠는가를 규정하기 위하여 선택된 항목

Char **where*, // *flag*를 가진 작업

Int *ntask*, // 출발실행가능한 복사개수

Int **tid* // 생성된 과제의 과제식별자보존

)

이 함수는 실행가능한 progm라는 ntask복사본을 만든다. Progm에 대한 인수는 argv로 지적된 배렬에 포함된다. 기발은 표 13-10에서 보여 준 선택값들의 합으로 설정된다.

표 13-10 pvm_spawn에서 flag파라메터

선택 기호	선택값	의미
PvmTask Default	0	호스트는 PVM에 의해 선택
PvmTaskHost	1	호스트는 어디에서나 지정
PvmTaskArch	2	구성방식은 어디에서나 지정
PvmTaskDebug	4	오유수정기밑에서 시동처리
PvmTaskTrace	8	PVM 추적자료가 발생된다
PvmMppFront	16	MPP앞단위에서 시동처리
PvmHost-Compl	32	호스트나머지모임리용

그룹화

집중조작을 지원하기 위하여 PVM은 libpvm3. a라고 부르는 개별적인 서고를 리용하는데 핵심부PVM서고우에 libpvm 3. a를 놓는다.

PVM데몬 pvmd는 그룹화함수를 조종하지 않으며 그룹봉사기라고 부르는 개별데몬에 의해 수행된다.

이 데몬은 첫번째 그룹함수가 호출될 때 자동적으로 출발한다. PVM은 동적그룹화를 지원한다. 과제는 임의의 순간에 그룹을 결합 혹은 분리할수 있다. 일부 그룹함수들을 표 13-11에 제시한다. PVM동적그룹화개념은 아주 유연적이다. 다중그룹이 될수 있고 과제는

표 13-11 PVM그룹함수

PVM 함수호출	의미
inum=pvm_joingroup("World");	호출과제는 그룹 World를 결합하고 실례수 inum에 할당한다 실례수는 MPI에서 렬과 같다
info=pvm_lvgroup("World");	호출과제는 그룹 World를 출발
tid=pvm_gettid("World", inum);	실례수로부터 과제 ID얻기
inum=pvm_getinst("World", tid);	과제 ID로부터 실례수얻기
gsize=pvm_gsize("World");	그룹크기얻기
info=pvm_barrier("World", 10);	호출과제는 World의 성원이 10이 될 때까지 차단(대기)
info=pvm_bcast("World", tag);	태그에 의해 식별되는 통보문을 World의 모든 성원들에 방송(자체는 제외)
info=pvm_reduce(...);	MPI에서 감소와 류사하다

동시에 각이한 그룹에 속할수 있다. 과제는 다른 그룹의 성원들에 통보함이 없이 임의의 시각에 그룹을 결합 혹은 탈퇴할수 있다. 과제가 언제나 그룹안에서 유일한 순위를 가지는 MPI와는 달리 PVM은 그것이 그룹을 탈퇴하거나 재결합할 때 각이한 경우의 수들이 배정될수 있다.

과제는 그룹의 성원이 아니다 하더라도 통보문을 그룹에 보낸다.

동적그룹화는 프로그람의 비결정동작을 증가시킨다. 실례로 과제가 그룹을 결합하거나 탈퇴할 때 방송동작은 각이한 결과를 가져 올수 있다. 과제는 방송통보문을 얻거나 얻지 못할수 있다. 장벽동작은 성원과제들이 그룹을 탈퇴한다면 교착상태에 이르게 할수 있다.

과제식별자

PVM은 가상기계안에서 과제, 그룹 혹은 pvmd를 주소화하기 위하여 32bit옹근수를 리용한다. 이 옹근수를 **과제식별자**(TID)라고 부른다. 일반적인 TID형식을 그림 3.17에 보여 준다.

32bit를 4개 마당으로 나눈다. 즉 봉사비트 S(데몬은 종종 봉사기라고 부른다.), 그룹비트 G, 12bit 호스트마당 H, 18bit국부마당 L.

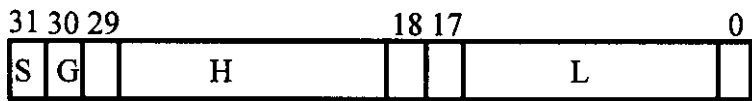



그림 13-7. PVM 일반 TID 형식

이 마당들의 여러가지 결합해석을 표 13-12에 제시한다. 여기서 $4095=2^{12}-1$은 가상기계에서 호스트의 최대수이다. 호스트당 과제의 최대수는 $262143=2^{18}-1$이다.

실지 실행에서 조작체계는 호스트당 과제수를 훨씬 작은 수로 제한한다.

S=G=0(즉 봉사기나 그룹이 없다.)일 때 TID는 과제를 식별한다. 과제는 H마당의 호스트에 상주하며 국부프로쎄스 ID는 L마당에 포함된다.

이 부호화도식에 따라 과제는 내부호스트통신이 없이 TID를 국부pvmd로 배정될수 있다.

표 13-12 과제식별자의 해석

S	G	H	L	의미
0	0	1..4095	1..262143	과제 ID
0	1	1..4095	Don't care	집단내방송주소
1	0	0	0	국부 pvmd
1	0	1..4095	0	A pvmd ID
1	1	작은 부수값		오유코드

S=0, G=1일 때 TID는 과제그룹을 식별하여 통보문을 내부방송으로 다중호스트에 보내게 한다.

이것은 집중 혹은 대역동작을 지원하는 과제그룹개념과 차이난다.

S=1, G=H=L=0일 때 TID는 국부pvmd를 식별한다. H마당이 정수이면 TID는 H마당에서 지적된 호스트수를 가지고 국부 혹은 원격pvmd를 식별한다. S=H=1일 때 TID는 오유조건을 지적하는데 쓴다.

13．3．3．PVM에서 통신

가상기계에 실행하는 PVM프로글마은데몬과 과제들을 포함한다. 데몬과 파제사이, 두개의 데몬사이 그리고 두개의 과제사이의 통신을 비롯한 여러 형태의 통신이 가능하다. 병렬계산을 위한 자료를 넘기기시키는것외에 PVMD들가운데서 heartheat통보문들을 넘기기시키는 통신이 요구된다.

통신규약 일반적인 PVM은 가상기계의 매 호스트 TCP와 UDP를 리용하는 다른 모든 호스트에 직접 련결할수 있다고 가정한다. 이 규약은 그것이 표준적이고 널리 리용할수 있기때문에 선택된다. 이것은 핵심부를 수정할 필요가 없다는 우점이 있지만 부가프로쎄스가 매우 크다. PVM에서 리용되는 통신규약을 그림 13-8에 보여 주었다. 그림에서 보는것처럼 세가지 형태의 통신 즉 두개의 pvmd사이, pvmd와 과제사이 그리고 두개의 과제사이의 통신이 있다.

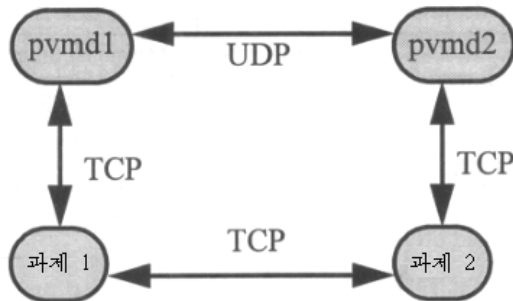



그림 13-8. 일반 PVM에서 리용되는 통신규약

두개의 pvmd사이의 통신에서는 UDP를 리용한다. TCP대신에 UDP를 리용하는 리유는 세가지이다. 첫번째 리유는 확대가능성이다. 매 TCP련결은 pvmd에서 파일해석기를 리용한다. 열려 지는 파일수는 임의의 조작체계에 의해 제한된다. 한편 단 하나의 UDP소케트는 많은 수의 원격UDP소케트와 통신할수 있다.

두번째 리유는 TCP를 리용할 때 n마디점가상기계에 대하여 $n(n-1)/2$개의 련결이 필요한것이다. 이것은 관리비용이 비싸다.

UDP는 다음과 같은 련결위상에 따라 초기화할수 있다. 즉 pvmd들사이에는 련결이 명시적으로 설정되지 말아야 한다.

세번째 리유는 호스트, pvmd 그리고 망고장을 검출하기 위하여 UDP에 시간경과를 설치하기가 더 쉽다는것이다. Pvmd와 과제들사이의 통신은 주로 TCP가 UDP에 비해 믿

음직한 전송을 보장하기때문에 TCP규약을 리용한다. 두개의 과제사이의 통신은 두가지 방식으로 달성될수 있다. 기정방식에 의하면 한 과제로부터 다른 과제에 전송되는 임의의 통보가 경로전체를 순회한다. 다른 방식은（원천과제 대 원천pvmd） 대 （목적pvmd 대 목적과제）이다.

교대적으로 원천과제로부터 목적과제로 TCP련결을 통해 통보문이 직접 전송될수 있다. 두번째 방법은 통보문순회부가처리는 감소시키지만 우에서 언급한 TCP련결의 약점을 가진다.

PVM통신기능 PVM의 사용자는 우에서 언급한 세가지 형태의 통신을 알 필요가 없다. 그것들은 단지 고수준통신기술의 묶음이다. PVM은 많은 기술들을 제공한 착상만을 례증하기 위하여 그중 몇가지만을 론의한다. 표본PVM통신기능을 그림 13-9에 보여 주었다.

때에 따라 PVM과제는 하나의 능동송신완충기와 하나의 능동수신완충기를 가진다. 한 과제로부터 다른 과제에로 자료를 보내기 위해서는 원천과제가 먼저 자료를 능동인 송신완충기에 통보문으로서 묶어야 하며 다음에 목적과제의 능동수신완충기에 보내기 위한 서고함수를 호출하여야 한다.

```
int bufid = pvm_initsend( int encoding )
int info = pvm_pkint(int *p, int nitem, int stride)
int info = pvm_send(int tid, int tag)
int info = pvm_mcast(int *tids, int ntasks, int tag)
int bufid = pvm_recv(int tid, int tag)
int bufid = pvm_nrecv(int tid, int tag)
int bufid = pvm_trecv(int tid, int tag, struct timeval *tmout)
int bufid = pvm_probe(int tid, int tag)
int bufid = pvm_bufinfo(int bufid, int *bytes, int *tag, int *tid)
int info = pvm_upkint(int *p, int nitem, int stride)
```

그림 13-9. PVM 통신기술의 표본

목적과제는 능동수신완충기에 통보문을 받기 위한 수신함수를 실행하여 다음통보문을 얻으려는 자료로 풀어 놓아야 한다. 송수신함수는 늘 능동인 완충기들을 리용한다는 것을 강조하므로 임의의 통보문완충기를 지정하지 않는다. 이것이 MPI와 다른 점이다.

함수 *PVM_initsend*는 새로운 송신완충기를 생성하며 그것을 현재의 능동완충기로 만들어 준다. *encoding*파라메터에서는 자료를 통보문으로 묶는데 리용될수 있는 자료형을 지정하며 그것은 아래의 PVM상수들중의 하나일수 있다.

- DVmData Default는 자료가 XDR(외부자료재현)형에 의해 부호화된다는것을 지적하며 그 형은 서로 다른 자료형을 리용하는 다른 종류의 마디들사이에서 자료교환에 리용되는 가동환경독립형이다.

- PvmDataRaw는 행자료가 어떤 부호화도 되지 않고 리용된다는것을 의미한다. 이것은 동종마디사이 통신에 적당하며 부호화의 무익성을 회피할수 있다.
- PvmDataInPlace는 보다 무효성을 줄일수 있는 세번째 방식을 의미한다. 그것은 실제 자료가 능동완충기에 복사되지 않고 단지 자료의 주소와 길이만이 복사되는것이다.

자료의 실제적인 묶기는 어떤 PVM함수 Pvmpk와 우의 자료형의 하나에 의해 진행된다. 실례로 100개 원소로 된 옹근수들 A(즉 A[0], A[2], …, A[98])의 매 첨수화된 요소를 능동송신완충기에 보장할 때 우리는 Pvm−Pkint(A, 50, 2)이라는 PVM함수에 접근할수 있는 데 여기서 A는 자료의 주소, 50은 자료원소의 개수, 2는 걸음이다.

다른 실례로 우리는 류점수형의 자료를 묶는데 Pvmpkfloal(…)를 리용할수 있다. PV<은 또한 서로 다른 형의 요소들을 포함하는 자료를 묶는 PVM−Packf(…)라는 함수로 제공한다. Pvm−send(tid, tag)함수는 능동송신완충기의 통보문을 tid에 의해 지적되는 과제에로 보내려고 통보선을 tag로서 꼬리 붙인다.

여러 과제에 통보문을 동시발송(방송)하기 위한 함수 Pvm−mcast(tids, ntasks, tag)는 과제의 계속이다. 함수 Pvm−recv와 Pvm−nrec는 각각 수신차단 및 수신비차단체계이다. tid와 tag는 둘 다 임의의 과제나 임의의 꼬리를 가리키는 −1이라는 총칭값을 가질수 있다.

실례 13.15. 실례 12. 1의 π계산을 위한 PVM프로그람

그림 13− 10에 π계산을 위한 PVM프로그람을 보여 주었다. 이 SPMD프로그람은 [0, 1]구간을 N개 령역으로 나누고 *n*개 과제를 리용한다. 매 과제는 N/n령역으로부터 부분합 mtpi를 계산하기 위한것이다. *n*부분합은 다음 감소연산에 의해 최종합으로 합쳐 진다. 원천코드는 Pic파일에 포함된다고 하자. 실행가능파일 PI는 아래의 명령을 리용하여 얻을수 있다.

즉

```
cc−I/PVM3/include pic livgpvm3. alibpvm3. alibpvm3. a−opi
```

사용자는 첫 과제를 기동하며 그것은 n−1개 과제로 새끼친다. 매 과제는 처음에는 명령

me=pvm_joingroup(“PI”);

을 실행한다.

그러면 이것은

pvm_initsend(PvmDataRaw);
pvm_plcint(&N,1,1);
pvm_mcast(tids,n−1,5);

을 실행한 자료에 의하여 자식과제에 대한 이 값들을 방출한다.

```
#define n16              /* number of tasks */
#include "pvm3.h"
main (int argc, char ** argv)
{
   int mytid, tids[n], me, i, N, rc, parent;
   double mypi, h, sum=0.0, x ;
   me = pvm_joingroup( "PI" );
   parent = pvm_parent();
   if (me == 0) {
        pvm_spawn("pi", (char**)0, 0, "", n-1, tids);
        printf("Enter the number of regions: ");
        scanf("%d",&N);
        pvm_initsend( PvmDataRaw );
        pvm_pkint(&N,1,1);
        pvm_mcast(tids,n-1,5);
   } else {
        pvm_recv(parent, 5);
        pvm_upkint(&N, 1, 1);
   }
   pvm_barrier("PI", n);    /* optional */
   h  = 1.0 / (double) N;
   for (i = me + 1; i <= N; i += n)  {
        x = h * ((double)i - 0.5);
        sum += 4.0 / (1.0 + x*x);
   }
   mypi = h * sum;
   pvm_reduce( PvmSum, &mypi, 1, PVM_DOUBLE, 6, "PI", 0);
   if ( me == 0 ) printf("pi is approximately  %.16f \n", mypi);
   pvm_lvgroup("PI");
   pvm_exit();
}
```

그림 13-10. π PVM 으로 씌여 진 프로그람

여기서 5는 임의의 통보문꼬리이다. 한편 매 자식과제는

pvm_recv(parent,5);/*pvm_mcast에서 그것을 정합시키는 꼬리표이다*/
pvm_upkint(&N,1,1);

을 실행함으로써 방송과 대등해 진다.

매 과제는 장벽

pvm_barrier(“PI”,n);/*그룹 PI에서 *n*개의 과제들은 동기화되여야 한다*/

을 실행하여 동기를 맞춘다.

다음의 행들은 바로 표준계산이다.

```
h=1.0/(double)N;
for(i=me+1;i<=N;i+=n){
    X=h*((double)i−0.5);
    Sum+=4.0/(1.0+x*x);
}
mypi=h*sum;
```

다음의 reduction연산은

```
pvm_reduce(PvmSum,&mypi,1,PVM_COUBLE,6,”PI”,0);
```

그룹 PI의 모든 과제의 감소합을 의미한다. 매 과제는 *mypi*위치에 PVM−DouBLE형의 한 자료항목을 부여 한다.

최종결과는 초기과제(위수가 0인)의 mypi에 기억되게 된다.

값 6은 또한 독자적인 통보문꼬리표이다.

보통 PVM감소는 다음의 함수원천을 가진다.

pvm_reduce(void(*func)(),void*buffer,int nitem,int datatype,int tag,char*group,int root);

결과를 인쇄한후 프로그람은

pvm_lvgroup(“PI”)and*pvm_exit*()

에 접근함으로써 지워 진다.

13. 4. 참고문헌주해와 련습문제

많은 통보문넘기기쏘프트웨어묶음을 포함하는 Cluster−Supporting쏘프트웨어에 대한 포괄적인 개괄을 [620]에서 주었다. 이런 쏘프트웨어는 1993년까지 리용되였다. 대표적인 통보문넘기기의 심오한 론의에 대하여서는 문헌 [118, 303, 372, 502, 574]에서 찾아 볼 수 있다.

개발된 모듈 및 확정병렬프로그람 그리고 포트란__M수법들은 문헌［141,247]에서 론의하였다.

PVM과 관련한 좋은 참고서로서는 문헌 [261]과 Sanderram의 론문 [597]들을 소개할 수 있다.

문헌 [375]에서는 PVM이 얼마나 효과적으로 실행되는가를 론의하였다. MPI에 대한 심오한 론의는 문헌 [283]과 [575]에서 주었다.

문헌 [112]와 [394]들은 MPI의 두가지 효과적인 실현실레를 주었다. 문헌 [322]와 [655]에서 저자들은 MPP에서 MPI성능에 대하여 체계적으로 서술하였다.

최신정보와 완벽한 참고자료들은 Web자원목록의 PVM, MPI홈페지에서 찾아 볼수 있다.

문 제

문제 13. 1. 아래의 MPI용어를 고찰하도록(실례 13.9에서) 첫 MPI−Send명령문을 간단히 정의하고 리용하시오.

(1) 통보문완충기
(2) 통보문봉투
(3) 통보문문맥

문제 13. 2. 실례 13.4참고.

다음의 매 코드가 실행된후 프로쎄스 2에서의 A[1]와 B[0]의 값은 얼마인가?

(1)
```
MPI_Alltoall(A,1,MPI_INT,B,1,MPI_INT,comm);
```

(2)
```
if(my_rank==0){
   MPI_Bcast(A,3,MPI_INT,root,comm);
   MPI_Send(B,3,MPI_INT,2,tag,comm);
}else if(my_rank==1){
   MPI_Send(B,3,MPI_INT,2,tag,comm);
   MPI_Bcast(A,3,MPI_INT,root,comm);
}else{
   MPI_Recv(B,3,MPI_INT,MPI_ANY_SOURCE,tag,comm);
   MPI_Bcast(A,3,MPI_INT,root,comm);
   MPI_Recv(B,3,MPI_INT,MPI_ANY_SOURCE,tag,comm);
}
```

(3)
```
if(my_rank==0){
   MPI_Send(A,3,MPI_INT,1,tag,comm);
   MPI_Recv(B,3,MPI_INT,2,tag,comm);
}else if(my_rank==1){
   MPI_Send(A,3,MPI_INT,2,tag,comm);
   MPI_Recv(B,3,MPI_INT,0,tag,comm);
}else{
   MPI_Isend(A,3,MPI_INT,0,tag,comm);
   MPI_Recv(B,3,MPI_INT,1,tag,comm);
}
```

(4)
```
if(my_rank==0){
```

```
    MPI_Isend(A,3,MPI_INT,1,tag,comm);
    MPI_Recv(B,3,MPI_INT,2,tag,comm);
}else if(my_rank==1){
    MPI_Isend(B,3,MPI_INT,2,tag,comm);
    MPI_Recv(B,3,MPI_INT,0,tag,comm);
}else{
    MPI_Send(B,3,MPI_INT,0,tag,comm);
    MPI_Recv(B,3,MPI_INT,1,tag,comm);
}
```

문제 13. 3. 7.2.1에서 은행문제를 풀기 위한 MPI프로그람을 작성하시오. 특히 출금(저금찾기) 및 예금거래를 허용하는 한개의 회계만 있다고 가정한다.

프로그람은 은행가과제와 두 주문자과제를 포함하고 있다. 은행가는 회계를 관리하며 매 주문자는 출금이나 예금거래를 초기화한다.

(1) 은행가와 주문자코드를 둘 다 쓰시오.

(2) 프로그람이 교착되지 않는가를 보시오.

(3) 프로그람이 어떤 주문자도 빈곤하지 않게 하는가를 보시오.

프로그람이 빈곤의 자유도를 보장할수 없다면 그 프로그람의 빈곤이 없게 될 필요충분조건을 만드시오

(4) 프로그람이 정확한 결과를 발생하는가를 보시오. 실례로 주문자가 출납으로부터 100$를 출금하려 할 때 은행가가 그 출납에서 100$를 덜고 그만한 량을 주문자에게 출금한다. 프로그람은 다른 손님이 그에게 거래를 요구할 때도 혼란되지 말아야 한다.

문제 13. 4. 야꼬비완화문제(실례 12. 4)를 위한 완성된(실행준비전) 병렬프로그람을 작성하기 위한 MPI를 리용한다. 골조코드는 그림 13− 11에 보여 주었다.

독자들은 아래의 과제를 수행하여야 한다.

(1) 잘 문서화되고 완성된 순차 및 병렬프로그람을 작성한다.

(2) 왜 변수나 명령문이 거기에 포함되는가를 코드의 형태로 론의하시오.

(3) 코드가 교착되지 않을것이라는것을 증명하시오.

(4) MPI병렬프로그람을 작성할 때 의뢰할수 없는것을 설명하면서 코드화의 명백성을 요구하시오.

이 련습문제로부터 구체적인 실례를 주어야 한다.

그렇게 할수 없으면 반대로 제시하시오.

```
float Arow[N], X[N], NewX, B, error, Temp ;
MPI_Comm comm ;
error = SomeLargeValue ;
initialize A, X, and B
MPI_Init (&argc, &argv) ;
comm = MPI_COMM_WORLD ;
MPI_Comm_rank (comm, &my_rank) ;
while ( error > SomeErrorBound ) {
     Temp = B;
     for ( i=0; i<N; i++) Temp -= Arow[i] * X[i] ;
     Temp /= A[my_rank] ;
     NewX = Temp + X[my_rank] ;
     MPI_Allgather (&NewX, 1, MPI_FLOAT, &X, 1, MPI_FLOAT, comm) ;
     Temp = Temp * Temp ;
     MPI_Allreduce (&Temp, &error, 1, MPI_FLOAT, MPI_SUM, comm) ;
}
```

그림 13-11. MPI 코드에서 야꼬비완화

문제 13.5. PVM을 리용하여 문제 13. 4를 반복하시오. 대응하는 MPI와 PVM코드사이에서 프로그람작성능력과 쏘프트웨어생산성에 대하여 평가하시오.

문제 13.6. 2개의 문제크기：N＝100000과 N=10000000을 가지는 계산을 위한 MPI로 SPMD프로그람을 작성하시오.

임의의 병렬콤퓨터로 코드를 개발하고 검사하시오. 프로그람은 1, 2, 3, 4, 5, 6, 7, 8개 처리기를 리용하여 최적화되고 실현될수 있다. 이 련습의 서술은 다음을 포함한다.

(1) 완성되고 잘 문서화된 순차 및 병렬프로그람
(2) 가동환경, 언어, 콤파일러선택, 서고들, 환경파라메터, 실행방식(묶음/호상작용, 바치다/공유)을 포함하여 실험환경과 수속의 표시에 걸리는 시간
(3) 순차실행시간, 병렬유한시간, 속도(mflop/s), 속도개선, 효률, 편리성을 포함하여 표화되고 계획화된 성능측정값
(4) 측정결과에 대한 해석. 기계크기와 문제크기를 초월하여 편리성과 규모변경성의 관측과 설명에 주의를 돌리시오.

문제 13.7. 문제 13.6을 PVM을 리용하여 반복하시오. MPI와 PVM코드사이 프로그람작성능력과 쏘프트웨어생산성에 대하여 평가하시오.

문제 13.8. 실례 12.2의 목표발견문제를 위한 완성된(실행준비된) 병렬프로그람을 쓰기 위하여 MPI사항을 리용한다. 골조코드는 그림 13-12에 제시하였다. 다음의 과제를 수행해야 한다.

(1) MPI코드를 가지고 완성되고 잘 문서화된 병렬프로그람을 작성하시오.
(2) 변수나 명령이 거기에 왜 있어야 하는가를 비롯하여 코드의 론리적단계를 설명하시오.
(3) 어떤 조건하에서 코드가 교착되지 않게 되는가를 보시오.
(4) MPI에서의 병렬프로그람작성에서 무엇이 좋고 무엇이 좋지 않는가를 설명함으로써 코드화경험을 개괄하시오. 이 련습에서 구체적인 실례를 준다. MPI의 리용에서 좋지 않은 점들이 있기때문에 그에 해당한 대응책을 제안하고 그것을 설명하시오.

```
#define     MaxTarget       10
#define     M               1000
#define     N               256
int  i, j, k = 0, group_size, my_rank;
float  A[M][N] ;
MPI_Init (&argc, &argv) ;
comm = MPI_COMM_WORLD ;
MPI_Comm_rank (comm, &my_rank) ;
compute A[*][my_rank]
j = my_rank ;
for ( i = 0 ; i < m ; i ++ )
   if ( IsTarget( A(i,j) ) ) {
      LocalTarget[k].direction = j ;
      LocalTarget[k].distance = i;
      k = k + 1 ;
      if ( k > MaxTarget ) break ;
   }
MPI_Op_create ( & target_reduce, Commute, & TargetReduce ) ;
MPI_Reduce(LocalTargets, GlobalTargets,
            TargetsType, TargetReduce, 0, comm);
MPI_Finalize() ;
```

그림 13-12. MPI를 리용하기 위한 목표발견코드

문제 13.9. 문제 13.8을 PVM을 리용하여 반복하시오. 같은 MPI와 PVM코드사이에서 프로그람작성능력과 쏘프트웨어생산성을 설명하시오.

문제 13.10. MPI와 PVM의 상대적인 우점과 결점을 보여 주는 큰 표를 편집하시오. 표는 중요한 통신함수를 초월하여 성능특징도 정통할수 있어야 한다.

표항목을 추리하여 론증하시오. 그에 대한 연구는 고준위로부터 저준위까지를 포함하여 여러가지 응용프로그람준위를 취급하게 될것이다.

전용프로그람에 해당한 문제 13.4로부터 13.9에서 얻은 프로그람비교결과들은 표에 있는 일반항목으로서의 공통속성들을 분류하도록 일반화할수 있다.

제１４장. 자료병렬프로그람작성

이 장에서는 사용자응용프로그람에서 자료병렬사용문제를 취급한다.

우리는 여기서 두가지 자료병렬프로그람작성언어인 Fortran 96과 고성능Fortran(HPF)을 구체적으로 론의한다. 이 프로그람작성환경은 수년간에 거쳐 병렬Fortran계산공통체에 의하여 연구개발된 결과이다.

Fortran 95, Fortran 2001과 HPF2와 같은 확장된 Fortran과 관련한 몇가지 내용들을 론의한다.

１４．１．자료병렬모형

자료병렬프로그람작성모형은 여러가지로 특징을 고찰할수 있다.

그중 일부는 공유변수와 통보문넘기기모형이 서로 다르다.

아래에서 그 차이점을 보자.

- **단일스레드화**

프로그람작성자의 관점에서 자료병렬프로그람은 조종의 단일스레드를 가지는 하나의 프로쎄스로 실행된다.

- **집합적자료구조우에서 병렬조작**

자료병렬프로그람의 단일걸음은 한 배렬의 서로 다른 요소들에 동일하게 적용되며 다중조작을 실행할수 있다.

- **성긴동기화**

이것은 모든 명령문이후의 암시적동기화이다.

이 명령문준위 동기화는 SIMD체계에서 고도로 동기화된것에 비할 때 분산적이다.

- **전역적인 이름공간**

모든 변수들은 유일한 주소공간에 존재한다. 모든 명령문들은 임의의 변수,대상을 접근할수 있다.

이것은 통보문통신과 상반된다. 여기에는 서로 다른 주소공간들의 변수가 있을수 있다.

- **암시적호상작용**

암시적장벽이 있으므로 자료병렬프로그람에서는 명시적표기화가 필요없다.

- **암시적자료확장**

프로그람작성자는 자료를 배정하기 위하여 어떻게 하여야 하는가를 명백히 밝힐 필요가 없다.

１４．２．Fortran 90 방법

인기 있는 자료병렬언어인 Fortran 77을 많이 개선한것이 Fortran 90이다. 우리는 병렬성을 지원하는 두가지 중요특징들만을 론의하기로 한다. 즉 병렬배렬조작, 배렬함수.

Intrinsic조작

모든 언어는 몇가지 Intrinsic연산(즉 +, −, *, /)과 함수(즉 sin, log)를 가진다. 이것들은 언어의 일부로 간주한다. 따라서 언어를 임의로 실행할 때 같은 의미들을 가지고 그것을 지원하여야 한다. 콤파일러는 사용자에 의한 정의와 규정을 더 진행하지 않고도 함수의 문법과 의미론을 알아 차린다. 반면에 Intrinsic함수를 일부 체계에 의하여 실행시킬수 있으나 다른 체계들에서는 실행시킬수 없다. Intrinsic함수는 서로 다른 실행에서 서로 다른 문법과 의미론을 가질수 있다.

실례로 C에서 print()함수는 그것이 모든 체계들에서 같은 문법과 의미론을 가지고 실행되므로 Intrinsic로 간주한다. 그러나 FFT에 대하여 C함수들은 Intrinsic이다.

１４．２．１． 병렬배렬조작

프로그람언어에서의 과학계산에 대한 강력한 지원을 하기 위하여 새로운 Fortran 90 표준은 프로그람언어의 배렬능력을 현저하게 확장하였다.

전체배렬, 배렬의 일부(배렬구역이라고도 한다.)가 조작대상으로 된다.

배렬구역

모든 Intrinsic함수들은 기본요소와 같이 동작하면서 전체배렬 혹은 배렬구역에 적용될수 있다. 또한 Intrinsic함수를 통하여 기본요소가 아닌 병렬배렬조작을 수행할수 있다. 이것은 14.2.2절에서 론의하였다.

실례 14.1. Fortran 90프로그람에서 배렬조종

다음과 같은 Fortran 90코드부분을 고찰하자

```
real, dimension(3,3)::A
real, dimension(2,2)::B,C

S1: C=0
S2: B=A(1:3:2,2:3)
S3: where((B−6)>0)C=sin(A(1:2,1:2))
```

주어 진 배렬 A에 대하여 우의 코드의 결과는 다음과 같다.

$$A=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}\quad B=\begin{bmatrix}2 & 3\\8 & 9\end{bmatrix}\quad C=\begin{bmatrix}0 & 0\\-0.7568 & -0.9589\end{bmatrix} \tag{14.1}$$

해석

자료병렬프로그람작성형식의 4가지 특성은 다음과 같은 실례코드를 가지고 실험할수 있다. 즉

- **단일스레드화와 성긴 동기화** 코드는 련이어(차례로) 만일 스레드들이 실행되는 3개의 명령 S1, S2, S3으로 구성되여 있다. 명령 S2(S3)은 S1(S2)의 모든 조작이 끝날 때까지 시작하지 않는다.
- **병렬배렬조작** 명령은 동시에 실행되는 배렬요소애서의 X연산을 포함할수 있다. 명령 S1은 배렬 C의 4개요소 모두에 0을 배정한다. 명령 S2은 배렬 A의 4개 요소들을 배렬 B에 배정한다. Where 명령 S3은 조건식 (B−6)>0에 따라 배렬 C를 배정한다. 조건이 진실인 C의 요소만이 수행된다. C의 첫행의 두 원소는 불변이다. 왜냐하면 조건이 거짓이기때문이다.
- **전역적이름공간** 모든 배렬변수와 요소들은 모든 명령문들에서 보인다. 프로쎄스전용자료의 개념은 없다.

배렬

배렬의 차원수를 **형태위수**(rank)라고 부른다. 실례 14.1에서 변수 A, B, C들은 모두 위수 2인 배렬들이다.

배렬구역은 매 배렬차원에 대하여 3원쌍(three−tuple) L:U:S(subscript triplet라고 한다.)는 배렬 A의 첫행, 둘째 행과 셋째 행에 의하여 형성된 배렬구역이다.

명령문에 여러개의 배렬 또는 배렬구역이 나타날 때에는 그것들은 적합하여야 한다. 실례로 배렬더하기 x+y에서 두 배렬은 같은 형태를 가져야 한다.

즉 그것들은 매 차원에서 같은 크기(즉 요소들의 개수)로 되여야 한다. 이와 같이 두 2×3 배렬들은 같은 형태를 가지지만 2×3배렬은 3×2배렬과 같은 형태를 가지지 않는다. 조작자가 스칼라 s와 배렬 X에 적용될 때 스칼라 s는 우선 모든 원소들이 s인 X와 같은 형태의 배렬로 확장된다. 이것을 실례14.1에서 볼수 있다. 여기서 B−6은 다음과 같은 방법으로 계산한다.

$$B-6=\begin{bmatrix}2 & 3\\8 & 9\end{bmatrix}-6=\begin{bmatrix}2 & 3\\8 & 9\end{bmatrix}-\begin{bmatrix}6 & 6\\6 & 6\end{bmatrix}=\begin{bmatrix}-4 & -3\\2 & 3\end{bmatrix} \tag{14.2}$$

1 4. 2. 2. Fortran 90에서 Intrinsic 함수

이상에서 언급한 기본요소와 같은 배렬조작외에도 Fortran 90은 배렬들을 전체적으로 조종하는 추가적인 Intrinsic함수를 제공한다. 우리는 아래에서 그 몇가지를 론의한다.

행렬연산

Fortran 90은 전체 배렬에 대하여 동작하는 여러개의 조작자들을 제공한다.

실례로 행렬 X는 transpose(X)를 호출하여 변환할수 있다.

한편 곱하기 함수 matmul은 $m \times n$ 형배렬 X를 $m \times p$ 형배렬 Y와 곱하여 $m \times p$형배렬로 귀환한다.

실례로 명령문 $Z = matmul(X, Y)$ 이다. 여기서 Z는 $m \times p$ 형배렬이여야 한다.

배렬구성

위수 1인 배렬은 반점으로 분리되고 "(/" 와(and)" /)"에 의하여 닫긴 값들의 렬인 배렬구성체를 통하여 형성될수 있다. 실례로 3차원벡토르 X는 값지정명령문 $X = (/1,3,5/)$ (또는 같은것이지만 $X = (/1:5:2/)$)에 의하여 구성할수 있다. 배렬을 생성하는 또 다른 방법은 여분의 차원을 추가하고 연산대상을 복사하는 방법으로 새로운 배렬을 구성하는 Intrinsic함수 확장을 리용하는것이다.

실례로 함수 spread((/1.3.5/),1.3)은 (/1.3.5/)를 3개복사하여 그것들을 1차원을 따라 확장한다. 함수 spread((11.3.51),2,2)로 (11.3.51)를 2개 복사하여 2차원으로 그것들을 확장한다. 그 결과들은 아래와 같다.

$$\text{spread}((/1,3,5/),1,3) = \begin{bmatrix} 1 & 3 & 5 \\ 1 & 3 & 5 \\ 1 & 3 & 5 \end{bmatrix}, \quad \text{spread}((/1,3,5/),2,) = \begin{bmatrix} 1 & 1 \\ 3 & 3 \\ 5 & 5 \end{bmatrix} \qquad (14.3)$$

표 14-1 Fortran 90으로의 감소함수들

함수	의미
SUM(X, DIM, MASK)	배렬 X의 원소들의 산수합
PRODUCT(X, DIM, MASK)	배렬 X의 원소들의 산수합
MAXVAL(X, DIM, MASK)	배렬 X의 원소들의 산수최대값
MINVAL(X, DIM, MASK)	배렬 X의 원소들의 산수최소값
ALL(X, DIM, MASK)	배렬 X의 원소들의 론리적(AND)
ANY(X, DIM, MASK)	배렬 X의 원소들의 론리합(OR)
COUNT(X, DIM, MASK)	배렬 X의 원소들의 수
MAXLOC(X, MASK)	배렬 X의 최대원소의 위치
MINLOC(X, MASK)	배렬 X의 최소원소의 위치

벡토르감소

Fortran 90은 표 14-1에 목록화한것과 같이 여러개의 Intrinsic함수들을 지원한다.

여기서 독립변수 DIM과 MAK들은 선택적이다.

우리는 아래에서 여러개의 실례들을 리용하여 이 함수들을 어떻게 리용하는가를 설명한다. 함수형태는

$$\text{Let } A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} \text{ Then SUM(A)=21, MACVAL(A)=6, and COUNT(A)=6}$$

와 같다.

독립변수 DIM은 주어 진 차원을 따라 감소를 속박하는데 리용된다.

독립변수 MASK는 MASK에 의하여 설정되는 조건이 진실인 그 원소들에 대한 감소를 속박한다.

A가 렬을 따라 합이 감소되므로 COUNT(A,A>2)이다. 2보다 더 큰 원소들이 4개이므로 5보다 더 작은 최대원소의 위치가 렬(colum) 1, 행(row) 2이므로 MAXLOC(A,A<5)=(2 1)이다.

배렬의 적합성은 병렬 Intrinsic함수들이 리용될 때 서로 다른 의미를 가진다. 실례로 두 행렬이 Intrinsic함수 Z=MALMUL(*X*,*Y*)를 리용하여 곱해질 때 적합성은 *X*의 두번째 차원이 *Y*의 첫번째 차원과 같은 크기를 가져야 한다는것을 의미한다.

따라서 *X*가 2×3배렬이고 *Y*가 3×2 배렬이면 *X*와 *Y*는 적합하다. 둘다 배렬이면 그것들은 적합하지 못하다.

계속하여 우리는 두 실례를 통하여 Fortran 90이 규칙적으로 구조화된 병렬계산응용프로그람에 대한 단순하면서도 깨끗한 프로그람들을 작성하기 쉽게 해준다는것을 알수 있다.

Fortran 90으로 프로그람을 작성하는 프로그람작성자는 콤파일러에 모든것을 맡겨 버림으로써 프로쎄스관리, 통신, 자료배치, 동기화항목에 대하여 걱정할 필요가 없다. Fortran 90은 Fortran 77보다 더 많은 특징들과 기능들을 제공해 준다. 일부 응용프로그람들에 대하여 Fortran 90프로그람은 대응하는 Fortran 77코드들보다 단순할수 있다.

실례 14.2. 야꼬비완화에 대한 Fortran 90 프로그람

야꼬비완화문제에 대한 Fortran 90프로그람을 그림 14-1에 제시하였다.

기본계산은 do while순환고리에 포함된다.

순환고리의 순차적이라는 본질은 순환걸음이 하나씩 실행된다는데 있다. 병렬계산은 매 순환안에서 진행될수 있다. Fortran 90의 배렬능력으로 하여 다만 세개의 명령문만이 요구된다. 명령문표시 S1, S2, S3들은 코드부분이 아니라는것을 주의해 준다. 그것들은 코드를 론의하는데 리용된다.

우리는 다음과 같은 몇가지를 관찰할수 있다.

- 코드는 간단하고 간결하며 알고리듬과 밀착되여 있다.
- Intrinsic연산자 SUM, MATMUL들을 리용하여 코드화를 단순하게 한다.

- Fortran 90은 간단히 론의한 HPF에 의하여 그리 유연하지 못하다.

```
parameter n = 1024
real A(n,n), x(n), b(n), error, Temp(n), DiagA(n)
integer i
error = SomeLargeValue
initialize A, x, and b
do i = 1, n
       DiagA(i) = A(i,i)
end do
do while ( error > ErrorBound )
S1:     Temp = ( b - MATMUL(A, x) ) / DiagA
S2:     x = Temp + x
S3:     error = SUM ( Temp * Temp )
end do
```

그림 14-1. Fortran 90에서 야꼬비완화문제

실례로 그림 14-1의 Fortran 90의 코드는 A의 대각선원소들을 배렬 DiagA를 얻는데 do순환고리를 요구한다.

실례 14.3. Fortran 90으로 작성된 가우스소거법

가우스소거법을 위한 Fortran 90의 코드를 그림 14-2에 제시하였다.

```
real A(n,n+1), x(n)
integer i, pivot_location(1)
do i = 1, n-1
   ! pivoting
   pivot_location = MAXLOC( ABS( A(i : n, i) ) )
   swap( A(i, i:n+1), A(i-1+pivot_location(1), i : n+1) )
   ! triangularization
   A(i, i:n+1) = A(i, i:n+1) / A(i,i)
   A(i+1:n, i+1:n+1) = A(i+1:n,i+1:n+1) - &
   SPREAD(A(i, i+1:n+1), 1, n-i) * SPREAD(A(i+1:n, i), 2, n-i+1)
end do
! back substitution
do i = n, 1, -1
   x(i) = A(i, n+1)
   A(1:i-1, n+1)  = A(1:i-1, n+1) - A(1:i-1, i)* x(i)
end do
```

그림 14-2. Fortran90에 의한 가우스소거법

코드는 두개의 do순환고리로 이루어 져 있다.

첫 순환고리는 대각선화를 실현하며 둘째 순환고리는 거꿀치환을 실현한다.

Fortran 90에 의한 가우스소거법에서 첫 순환고리의 매 순환 *i*에서 원소 “Pivot”의 위치가 처음에는 큰 절대값을 가지고 얻어 진다.

교체 부분루틴은 Pivot행을 현재행 *i*와 교체된다. 다음으로 *i*째 행아래의 *i*째 렬의 모든 A의 원소들이 모두 령으로 된다.

１４．３．고성능 Fortran

고성능 Fortran(HPF)은 다음과 같은 목적을 달성하기 위하여 Fortran 90을 확장하여 설계한 표준화된 언어이다[370, 419].

- 자료병렬프로그람작성
- 비단일 기억접근비용을 가진 MIMD와 SIMD우에서의 성능을 높이는것
- 여러가지 구성방식을 위한 코드조종능력

첫째 목적을 달성하기 위하여 HDF는 FORALL구성체, INDEPENDENT명령, 일부 추가적 Intrinsic함수들을 도입하였다.

이 새로운 언어특징들은 HPF가 Fortran 90보다 더 일반적은 배렬구역들과 병렬계산패턴들을 밝히는데서 유연하게 한다.

둘째 목적을 달성하기 위하여 HPF는 ALIGN과 DISTIBUTE와 같은 명령들을 제공한다.

이 명령들은 사용자가 콤파일러에 프로쎄스들사이에 자료가 어떻게 배치되여야 하는가를 지적하여 통신무효시간을 최소화하고 작업량이 고르롭게 분배되도록 한다.

HPF는 또한 사용자가 특수한 구성방식적인 낮은 준위환경을 리용하게 하는 EXTRINSIC수속과 같은 세번째 목적달성을 지향한 특징들을 가진다.

이 절에서 우리는 가장 중요한 첫 두가지 목적과 관련한 HPF의 특징들을 론의한다.

우리는 기본적인 개념을 달성하기 위하여 몇가지 간단한 실례를 들기로 한다.

１４．３．１．자료병렬성을 위한 지원

HPF는 사용자가 자료병렬성을 규정하기 위한 4가지 메하니즘을 지원한다. 그것들은 배렬표현식과 값지정명령문, 배렬Intrinsic함수, FORALL명령문, INDEPENDENT명령이다. 이것들중에서 첫 두개는 이미 Fortran 90에서 리용하고 있으며 HPF는 더 많은 Intrinsic를 추가한다. FORALL과 INDEPENDENT는 새로운 특징들이다.

FORALL구성체

HPF의 FORALL명령문의 Fortran 90의 배렬값지정명령문과 류사하나 더 유연하다.

다음 실례코드를 고찰하자.

```
FORALL(i=2.5,X(i)>0)X(i)=X(i−1)+X(i+1)
```

“I=2.5”부분을 HPF에서 호랄 트리플리트 스페크(*forall triplet spec*)라고 부른다.

여기서 i는 첨수변수이다.

subscript triplet 2:5는 아래 한계 2, 웃한계 5, 그리고 1의 default stride인 2:5:1과 등가이다. foral triplet spec는 유효첨수값모임 {2,3,4,5}를 결정한다.

X(i)>0부분은 스칼라값을 가지는 표현식이며 mask라고 부른다. 능동첨수값모임은 mask가 진실인 그런 유효첨수값들의 부분모임이다.

처음에 FORALL명령문에서 X=[1, −1, 2, −2, 3, −3]이라고 가정하자.

그러면 X(2)=−1<0 이고 X(4)=−2<0이므로 능동모임은 {3,5}이다.

능동함수값모임이 결정된 다음에 값지정명령문이 모든 표현식들은 모든 능동첨수값 3，5에 대하여 임의의 순서로 동시에 계산된다.

```
X(3−1)+X(3+1)evaluates to−1+(−2)=−3
X(5−1)+X(5+1)evaluates to−2+(−3)=−5
```

그러면 모든 능동첨수값(3과 5)에 대하여 왼쪽배렬요소들은 대응하는 오른쪽값들로 동시에 값이 배렬된다.

왼쪽배렬의 나머지 요소들은 불변이다.

따라서 우의 FORALL이 실행된 후에 배렬 X의 결과는 X[1, −1, 3, −2, −5, −3]이다.

FIORALL명령문에서는 한개이상의 forall−triplet spec들이 있을수 있다. 그러면 함수값들이 결합되여 리용된다. 실례로 FORALL명령문

FORALL(i=1:2,j=1:3,Y(i,j)>0)Z(i,j)=1/Y(i,j)은 Fortran 90명령문
where(Y(1:2,1:2)>0)Z(1:2,1:3)=1/Y(1:2,1:3)

과 등가이다.

첨수값들의 유효한 결합모임은 모임 ｛(1,1)(1,2)(1,3)(2,1)(2,2)(2,3)｝이며 능동결합모임은 Y(I,j)가 0보다 큰 부분모임이다.

많은 병렬계산들은 FORALL을 리용하여 쉽게 규정할수 있지만 Fortran 90에서는 그렇지 못하다.

FORALL명령문

FORALL(I=1:4,j=1:5,k=1:6)X(I,j,k)=k+j−k
은 다음의 Fortran 90 배렬값지정명령문에 의하여 동등하게 표현할수 있다. 즉

```
X=SPREAD(SPREAD((/1:4), 2, 5), 3, 6)
```

```
&          + SPREAD(SPREAD((/1:5), 1, 4), 3, 6)
&          – SPREAD(SPREAD((/1:6), 1, 4), 2, 5)
```

Fortran 90코드가 리해하고 효과적으로 실행하는데 훨씬 더 어렵다는것을 주의하시오.
또 다른 실례로 간단한 FORALL명령문

```
FORALL(i=1:n)X(i,j(i))=Y(i)
```

은 Fortran 90에서 한개의 값지정명령문으로 표현할수 없다. 때때로 사용자는 FORALL 명령문에서 여러개의 값지정명령문을 가질것을 요구할수 있다. 이것은 다음의 코드로서 실행하는바와 같이 FORALL구성체 혹은 다중명령문 FORALL이라고 부르는 더 일반적인 형태의 FORALL명령문으로서 수행할수 있다.

```
FORALL(i=1:n)
    A(i)=sin(B(i))
    C(i)=sqrt(A(i)*A(i))
    D(i)=B(i)+2
END FORALL
```

둘째 값지정명령문은 첫 값지정명령문에서의 계산(여기서 sin함수를 *n*번계산한것과 같다)들이 모두 끝난 다음에야 비로소 시작된다. 따라서 둘째값지정명령문에서 리용된 배렬 A는 첫 값지정명령문에서 계산된 새로운 값을 가진다.

류사한 방법으로 세번째 명령문은 그것이 두번째 값지정명령문에서 계산된 배렬 c를 리용하는것 같지 않지만 두번째가 끝난 다음에야 시작된다. C가 B의 가명(EQUIVALENT를 거쳐)이 될수 있다.

그러나 배렬 B는 두번째 값지정명령문에서 계산된 C의 새로운 값을 리용한다.

우리는 함수와 수속들이 FORALL명령문에서 접근될수 있다는데 주목한다. 유일한 요구는 그것들이 순수해야 한다는것 즉 부작용에 무관계해야 한다는것이다. HPF는 콤파일러가 함수 혹은 수속이 순수한가 아닌가를 결정하는데 도움을 주는 언어적인 속박 목록을 제공해 준다.

실례로 이러한 한가지 속박은 어떤 전역변수가 값지정명령문의 왼변에 나타나지 않는다는것이다.

독립인 명령들

프로그람작성자는 이 명령들을 리용하여 순환고리에 반복들가운데서 고리이행의존성이 존재하지 않는다는것을 콤파일러에 선언한다. 다시말하여 모든 반복들은 독립적으로 실행될수 있다.

실례 14.4. INDEPENDENT 명령

다음 코드에서 INDEPENDENT명령의 효과는 그림 14-3에서 해설하였다. 다음의 코

드들에서 화살표는 실행순서, 속박조건을 자리킨다.

```
!HPF$ INDEPENDENT                    !HPF$ INDEPENDENT
   FORALL(i=1:2)                          do i=1,2
      A(i)=foo(B(i))                        A(i)=foo(B(i))
      C(D(i))=log(E(i))                     C(D(i))=log(E(i))
   END FORALL                             end do
```

그림 14-3 ㄱ)를 고찰하자.

여기서 INDEPENDENT명령은 FORALL명령문에 존재하지 않는다. 성긴동기화때문에 첫 명령문 $A(i) = f_{00}(B(i))$ 의 실례들은 둘다 두번째 명령문 $C(P(i)) = \log(E(i))$ 의 임의의것이 시작되기전에 끝나야 한다. 그러나 FORALL은 매 명령문의 두 실례가 다 병렬로 실행될수 있다는것을 가정한다. 우리는 두 명령문사이의 암시적장벽을 시작해 볼수 있다.

INDEPENDENT명령이 추가되면 콤파일러는 장벽동기화이 제거될수 있다는것을 알려준다. 따라서 우리는 그림 14-3 ㄴ)를 얻는다. 콤파일러가 부분루틴 foo()이 순수한지 또 가명이 존재하는지 알수 없으므로 INDEPENDENT명령이 도움이 없이는 이런 최량화를 수행할수 없다는것에 주의해야 한다.

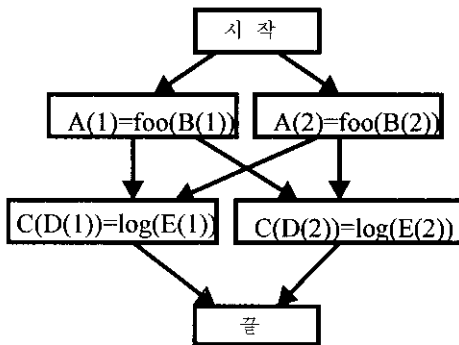
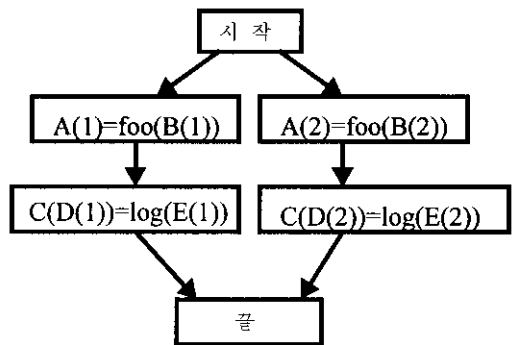
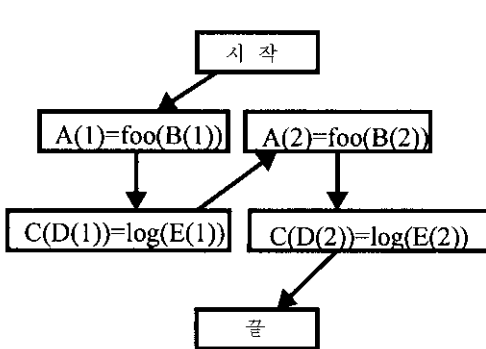
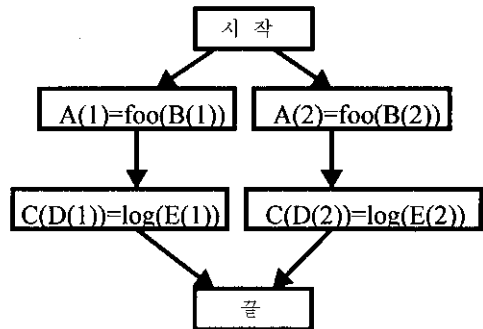



그림 14-3. INDEPENDENT명령의 도식

DO순환고리(그림 14−3 ㄷ)는 각이한 순서에 따르며 2개의 반복을 순차적으로 실행한다. 그러나 INDEPENDENT명령이 추가되면 (그림 14−3 ㄹ)콤파일러는 그것을 최량화하여 INDEPENDENT를 가진 FORALL명령문과 꼭 같이 동작할수 있게 한다.

14. 3. 2. HPF 에서의 자료넘기기

자료넘기기는 두가지 목적을 달성하려는 프로쎄스들에 대한 자료배치를 의미한다. 그 두가지 목적이라는것은 다음과 같다.

- 프로쎄스들사이에 통신을 최소화하는것.
- 작업량이 리용가치가 있는 프로쎄스들사이에 평등분포되도록 하는것

HPF콤파일러는 OWner−compute rule을 리용하여 작업량을 분배할수 있다. 즉 자료항목과 관련된 계산은 그 자료항목을 가진 프로쎄스로 수행한다. 따라서 자료넘기기는 간접적으로 작업량분포를 결정한다.

HPF는 콤파일러가 마디들에 자료를 가장 합리적으로 넘기도록 권고할수 있게 프로그람작성자가 리용할수 있는 명령들을 제공한다.

우리는 HPF자료넘기기를 설명할수 있는 또 하나의 중요한 명령을 보여 주는 여러가지 실례를 리용한다.

실례 14.5 HPF프로그람토막에서 자료넘기기 HPF코드부분을 고찰하자.

Consider the following HPF code Fragment:

```
          Integer  A(100),B(100),C(101),I
!HPF$  ALIGN  A(i)  WITH  B(i−1)
!HPF$  PROCESSOR  N(4)
!HPF$  DISTRIBUTE  A(BLOCK)  ONTO  N
!HPF$  DISTRIBUTE  C(CYCLIC)  ONTO  N
          FORALL  (i=2:100)
                A(i)=A(i)+B(i−1)
                C(i)=C(i−1)+C(i)+C(i+1)
          END  FORALL
```

그림 14−4에 제시된것이 바로 우의 코드이다.

이 코드에 의하여 자료넘기기가 실현된다. HPF에서 자료넘기기는 두개의 프로쎄스 즉 론리적넘기기와 물리적넘기기로 실현된다.

론리적넘기기는 여러개의 가상마디(추상 처리기)들에 자료를 넘기는 콤파일러명령을 통하여 사용자가 규정한다.

물리적넘기기는 실질적인 콤파일러에서 이 가상마디들을 물리적마디(처리기)들에 넘

긴다. 물리적넘기기는 체계에 의하여 진행되며 사용자에게는 명백하다.

가상적마디는 한개의 물리적마디에만 넘겨 진다. 다시 말하여 같은 가상마디에 배치된 모든 자료들은 같은 물리적마디들에 넘겨 져야 한다. 론리적넘기기는 두 단계로 구성되는바 그것은 배렬과 분배이다.

우의 코드와 련관시켜 보면 PROCESSOR명령은 다음과 같다.

```
!HPF$ PROCESSOR N(4)
```

콤파일러에게 그림 14-4에 제시된 4개의 가상마디 N1, N2, N3, N4의 선형적인 배렬

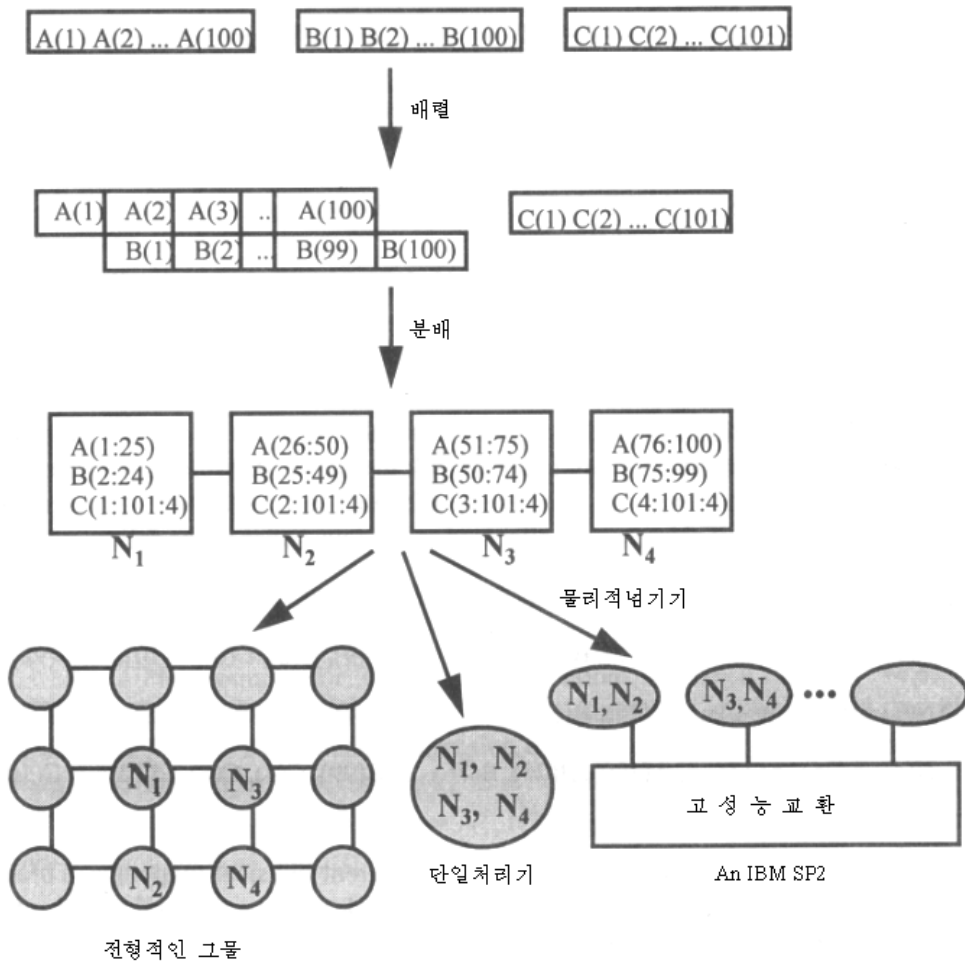



그림 14-4. HPF 자료배렬을 리용한 프로그람작성순차적 및 병렬 처리기들에서 자료넘기기

을 이 응용프로그람이 리용하기 바란다는것을 알려 준다. 이 마디의 배렬의 이름은 N이다.

HPF는 가상마디모임의 임의의 직선배렬로써 정의할수 있도록 한다.
실례로 콤파일러명령

```
!HPF$ PROCESSOR N(4, 5)
```

는 20개마디들이 4×5개의 쪼각으로 배렬되도록 하며 명령

```
!HPF$ PROCESSOR N(4, 5, 6)
```

은 120개의 마디들을 4*5*6개의 쪼각으로 배렬되도록 한다.
실례 14.5는 세가지 자료배렬 A, B, C들을 리용한다.
한줄배렬단계는 다만 하나의 ALIGN에서 명령

```
!HPF$ ALIGN A(i) WITH B(I−1)
```

만으로 이루어 진다.
이 명령은 콤파일러에 A(i)와 B(i−1)이 같은 마디에 넘겨 져야 한다는것을 알려 준다. 그러나 그것이 어느 마디인가를 알려 주지 못한다.
배치된 자료요소들은 프로쎄스들사이의 통신을 줄이기 위하여 같은 프로쎄스에 기억된다.
허위변수 *i*의 령역은 한개의 명령으로 제한된다. 그것은 임의의 변수에 의하여 교체될수 있다.
배렬 C가 어떻게 정렬되여야 하는가를 가리키는 명령은 없다는데 주의하여야 한다.
결국 이것은 임의의 다른 배렬들로 정렬되지 않는다.
분배단계에서는 프로그람작성자가 두개의 DISTRBUTE명령들을 리용한다.

```
!HPF$ DISTRIBUTE A(BLOCK) ONTO N
!HPF$DISTRIBUTE C(CYCLIC) ONTO N
```

마디배렬에 A와 B가 블로크형태로 분포되여야 하며 C는 주기적인 형태로 분포되여야 한다는것을 권고한다.
첫 DISTRIBUTE명령에서는 배렬 A만 나타났지만 B가 A와 함께 정렬되였으므로 그것은 배렬 B에도 자연히 분포된다. 블로크분포는 배렬을 여러개의 련쇄적인 배렬요소들의 묶음(블로크)으로 나누어 한 블로크를 한개 마디에 배치한다.
주기적인 분포는 고르롭게 나누어 모든 *i*째 마디에 배치되도록 한다.
실례로 실례 14.5에서 배렬요소들은 다음과 같은 방법으로 4개의 마디들에 배치된다. 즉

N(1):A(1),A(2),⋯,A(25),B(1),B(2),⋯,B(24),C(1),C(5),C(9),⋯,C(97),C(101)
N(2):A(26),A(27),⋯,A(50),B(25),B(26),⋯,B(49),C(2),C(6),C(10),⋯,C(98)
N(3):A(51),A(52),⋯,A(75),B(50),B(51),⋯,B(74),C(3),C(7),C(11),⋯,C(99)
N(4):A(76),A(77),⋯,A(100),B(75),B(76),⋯,B(99),C(4),C(8),C(12),⋯,C(9100)

같은 론리적넘기기를 서로 다른 물리적넘기기에서 리용할수 있다.
그림 14−4에 이와 같은 3가지 넘기기를 제시하였다.
4개의 가상적마디들은 쪼각위상기하학적구조를 가지는 Intel peragon콤퓨터의 4개의 물리적마디들에 넘길수 있다.
같은 가상마디들이 다단계 호상련결을 리용하는 IBM 물리적마디들에 넘겨 질수 있다.
사실상 HPF코드는 단일 프로쎄스에 의하여 실행될수 있는바 여기서 모든 가상마디들은 같은 물리적프로쎄스에 넘겨 진다.
HPF의 우점은 변화시키지 않으면서 같은 코드(모든 명령들과 함께)와 모두 세개의 콤퓨터에서 리용될수 있다는것이다.
매 체계는 자기의 물리적넘기기를 관리한다.
실례 14.5의 코드에서 얼마나 많은 통신이 요구되는가를 보자.
값지정명령문 A(i)=A(i)+B(I−1)은 A(i)를 B(i−1)과 결합하는 방법으로 어떤 통신도 요구하지 않는다. 그러나 값지정명령문 C(i)=C(I−1)+C(i)+C(I+1)은 주기적인 분포가 서로 다른 마디들에 C(I−1), C(i), C(i+1)들을 배치하므로 두 배렬이 요소들을 통신할것을 요구한다.
이것은 총체적으로 약 200개의 요소들이 통신되도록 한다.
따라서 주기적분포는 좋지 못한 선택이다. 우리는 통신을 줄이기 위하여 블로크분포 코드를 리용할수 있다.

```
!HPF$ DISTRIBUTE C(BLOCK) ONTON
```

이때 배렬 C는 다음과 같이 분포된다.

N1: C(1),C(2),⋯,C(26)
N2: C(27),C(28),⋯,C(52)
N3: C(53),C(54),⋯,C(78)
N4: C(79),C(80),⋯,C(101)

마디 N4는 불과 23개요소들만으로 배치된다.
이제 매 블로크의 경계요소들만을 통신할 필요가 제기된다. 즉 총 6개의 요소 C(26), C(27), C(52), C(53), C(78), C(79)들을 통신하여야 한다.
이것은 HPF의 또 다른 우점을 보여 준다. 즉 실행가능한 명령문이든 선언이든 변화시킬 필요가 없다.
성능을 개선하기 위하여 필요한 모든것이 콤파일러명령을 변화시키는것이다.

실례 14.6. 가우스소거법을 위한 HPF 프로그람

이 HPF프로그람은 그림 14-5에 제시하였다.

DISTRIBUTE명령은 A의 첫번째 차원이 블로크에 분포되여야 한다는것을 말해 준다.

A의 둘째 차원에서 *은 실패라는것을 의미한다.

이것은 배렬토막 $A(i,1:n)$이 같은 마디에 넘겨 진다는것을 의미한다.

4개의 가상마디들이 있으며 n은 4로 나누어 진다고 가정하자. 그러면 자료분포는 다음과 같다. 즉

```
parameter n = 32
real A(n,n+1), x(n)
integer i, pivot_location(1)
!HPF$ PROCESSOR Nodes(4)
!HPF$ ALIGN x(i) WITH A(i,j)
!HPF$ DISTRIBUTE A(BLOCK,*) ONTO Nodes
do i = 1, n-1
   ! pivoting
   pivot_location = MAXLOC( ABS( A(i : n, i) ) )
   swap( A(i, i:n+1), A(i-1+pivot_location(1), i : n+1) )
   ! triangularization
   A(i, i:n+1) = A(i, i:n+1) / A(i,i)
   FORALL ( j = i+1 : n, k = i+1 : n+1 ) A(j, k) = A(j, k) - A(j, i) * A(i, k)
end do
! back substitution
do i = n, 1, -1
   x(i) = A(i, n+1)
   A(1:i-1, n+1)  = A(1:i-1, n+1) - A(1:i-1, i)* x(i)
end do
```

그림 14-5. HPF의 가우스소거법

N1:A(1:n/4,1:n),b(1:n/4),x(1:n)

N2:A(n/4+1:2*n/4,1:n),b(n/4+1:2*n/4),Temp(n/4+1:2*n/4),x(1:n)

N3:A(2*n/4+1:3*n/4,1:n),b(2*n/4+1:3*n/4),Temp(2*n/4+1:3*n/4),x(1:n)

N4:A(3*n/4+1:n,1:n),b(3*n/4+1:n),Temp(3*n/4+1:n),x(1:n)

대응하는 자료분포는 표 14-2에 제시되였다.

이 분포에서 문제는 부하불균형성이다. 대각선화단계를 고찰하자.

8번째 반복이후에 마디 1은 놀고 있으며 16번 반복후에는 마디 1과 2가 놀고 있다.

작업은 자료가 주기적분포를 따르도록 함으로써 균형화되는바 이것은 련습으로 남긴다.

표 14-2 4개의 마디에 따르는 자료분포

마디 1	A(1:8, 1:33), x(1:8)
마디 2	A(9:16, 1:33), x(9:16)
마디 3	A(17:24, 1:33), x(17:24)
마디 4	A(25:32, 1:33), x(25:32)

14. 3. 3. Fortran 90과 HPF 개괄

우리는 아래에서 사용자응용프로그람에서 자료병렬을 리용하여 Fortran 90 또는 HPF를 리용하는데서 나서는 일련의 실제적인 문제점들을 요약한다.

병렬문제

단일스레드로 하여 자료병렬프로그람에서는 다중마디들이 서로 다른 자료기지부분상에서 같은 프로그람을 실행할수 있지만 론리적으로는 프로쎄스이다. 대다수 병렬문제들은 체계가 관리한다. 사용자는 프로쎄스들을 만들고 없애며 그룹을 형성하는데 대하여 걱정할 필요가 없다.

얼마나 많은 프로쎄스들이 프로그람을 가동시키는가를 알 필요가 없다.

명령급자료병렬과 그것이 구성되는 개소에서는 동시에 서로 다른 명령들이 수행되도록 서로 다른 마디들을 허용해 준다.

따라서 SMD와 SPMD계산이 모두 지원되지만 MPMD는 지원되지 않는다.

각이한 배렬과제가 사용되거나 또는 각이한 FORALL명령문이 실행될 때 프로그람의 병렬정도는 동적으로 변한다.

호상작용문제

약한 동기로 하여 거의 모든 호상작용문제는 자료병렬프로그람에서 제껴놓는다.

Fortran 90과 HPF에서 호상작용조작은 없다.

Fortran 90은 소거기능지원, 축소기능지원을 위한 9가지 기능을 보장하지만 사건고정, 아래로 찾기 또는 추출을 지원하지 못한다.

이것은 또한 사용자가 정한 축소조작도 지원하지 않는다.

HPF는 서로 루틴을 통하여 Fortran 90의 능력을 확장한다. 여기에는 추가적인 축소기능과 입력기능, 찾기기능이 포함된다. 자료병렬프로그람에서 호상작용은 협력적이다.

여기에는 하나의 호상작용방식만이 있다. 즉 명령급동기이다.

한개 명령문의 모든 조직은 다음 명령문의 조작이 시작되기전에 끝나야 한다. 통신은 배렬과제를 통하여 절대적이다. 정상적인 및 비정상적인 통신형식은 아래에서 제시된 서로 다른 배렬첨수들에 의하여 지정될수 있다.

```
!HPF$ ALIGN A(i) WITH B(i)
A(i)=B(i-1)  ! 정규적인 왼쪽이동
A(V(i))B(i)  ! 비정규통신, 패턴은 첨수배렬 V를 통하여 정의된다.
```

HPF는 자료위치를 찾음으로써 통신부가프로쎄스를 최소화하기 위한 배렬넘기기명령모임을 보장한다.

가장 중요한 명령들은 PROCESSOR,ALIGN,DISTRIBUTE들이다.

HPF는 또한 통신패턴을 변경시키기 위한 동적인 REALIGN과 REDISTRIBUTE명령문들을 제공한다.

이 자료넘기기구성체는 통신패턴들이 정규적일 때 보다 효과적이다.

의미론적문제들

단일스레드화와 속박 없는 동기화로 하여 자료병렬프로그람은 완전히 의미론적인것으로 된다.

그것은 교착 또는 교체를 가질수 없다.

다만 순차적인 프로그람에서와 같은 리유로 순환이 끝나지 않은것으로 인한 비완료만이 있을수 있다. Fortran 90과 HPF자료병렬프로그람은 순차적인 Fortran 90 프로그람과 구조화, 정확성, 완전성, 결합성문제에서 류사하다.

여기에는 병렬성에 의하여 초래되는 추가적인 복잡성이 없다.

자료병렬프로그람은 단일배정규칙이 만족되기만 하면 확정적이다. 임의의 배렬요소는 명령문에서 자주 대입한다. 단일배정규칙을 만족시키지 못하는 임의의 프로그람은 Fortran 90 또는 HPF에서 틀린것으로 간주한다.

실례 14.7. 자료병렬성에서 단일배정

```
Consider  A(5),I(5),J(5)
S1:    I=(/1,2,3,4,5/)
S2:    J=(/1,2,2,4,5/)
S3:    A(I)=I+2
S4:    A(2:4)=A(1:3)+A(3:5)
S5:    A(J)=J
```

아래의 코드단락(부분)을 HPF표기로 고찰하자.

병렬 I의 모든 요소가 서로 다르므로 단일배정은 A=(/3,4,5,6,7/)를 생성하는 명령문 S3에 의하여 만족된다.

대조적으로 명령문 S5가 A(2)에 두번 대입되였다는것을 의미하는 J(2)=J(3)때문에 틀린것으로 된다.

S4가 FORALL(k=2:4) A(k)=A(k−1)+A(k+1)와 등가라는데 대하여 주의해야 한다.

이것은 순차적순환고리

```
do  k=2,  4
        A(k)=A(k−1)+A(k+1)
End  do
```

와 등가가 아니다.

S4에서 3개의 오른쪽 표현식 모두가 배렬 A의 현재값을 리용하여 임의의 순서로 평

가된다. A(1)+A(3)=3+5+=8, A(2)+A(4)=10, A(3)+A(5)=12이다.

그러면 이 세개의 값은 다시 임의의 순서로 A(2) ,A(3) ,A(5)에 대입된다. 그렇지만 S4후에 A(/3,8,10,12,7/)이 실행된다.

프로그람작성능력

Fortran 90과 HPF가 고수준언어표준형인것으로 하여 임의의 정합프로그람은 좋은 이식성을 가진다. 대부분의 병렬성, 대화성, 의미론적문제점이 체계에 의하여 실현되는것과 그 자체의 단순한 의미론적특성으로 하여 Fortran 90과 HPF는 리용에서 편리한것으로 된다.

왜 실용적인 병렬프로그람작성이 우위를 차지하지 못하는가?

왜 여전히 사람들은 다른 수법을 따르고 리용하겠는가?

그 주되는 원인은 현재의 Fortran 90과 HPF언어정의가 일반성과 효률성에서 여러가지 제한성을 가지기때문이다.

Fortran 90과 HPF는 조종병렬성의 탐색을 지원하지 못한다.

이 언어들은 비동기반복, 작업대기, 관흐름 혹은 자료기지와 같은 알고리듬적파라다임들, 자료기지와 같은 응용프로그람들을 지원하는데 적합하지 못하다.

실례로 그것은 7.2.1절에서 고찰한 은행문제라든가 12.1.3절에서 배운 목표탐색과 같은 문제들을 위한 효과적인 자료병렬코드를 이상의 언어로 해결하기 어렵다.

HPF자료대입명령은 다만 정규적인 통신패턴으로 배렬을 지원할수 밖에 없다. 이것은 일반자료구조나 비정규통신패턴으로 병렬알고리듬작성을 효과적으로 지원할수 있겠는지 명백하지 못하다는 생각을 가지게 한다. 또한 이렇게 의심하는것이 옳지 않다고 단정할수 없다.

실례 14.8. GPF에서 목표탐색

아래에 HPF목표탐색코드의 기본줄거리가 제시되였다.

```
complex A(N,M)
integer temp1(N,M) temp2(N,M)
integer direction(Max Targets), distance(Max Target)
integer I,j
L1: FORALL(I=1:N,j=1:M)temp1(I,j)=IsTarget(A(I,j))
L2: temp2=SUM_PREFIX(temp1,MASK=(temp1>0))
L3: FORALL(I=1:N,j=1:M;temps(I,j)>0 and temp2(I,j)<=MaxTargets)
        distance(temp2(I,j))=I
        direction(temp2(I,j))=j
    END FORALL
```

FORALL명령문 L1은 배렬 A의 매 요소들을 동시에 평가하며 A(i,j)가 목표이면 temp1(ij)=1를 대입한다. 4개의 목표 A(1,), A(1,4), A(4,4)가 있다고 하자. 이때 temp1은

아래와 같은 값을 가진다.

$$temp1 = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad temp2 = sum_prefix(temp1) = \begin{bmatrix} 0 & 0 & 2 & 3 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix} \quad (14.4)$$

L2은 temp1의 모든 마디의 요소들의 전치합배렬 temp2에 대입한다. 다른 FORALL은 목표목록(거리배렬과 방향배렬에 의하여 표시되는)을 갱신한다.

Temp2의 모든 비령인 원소가 거리이므로 우에서 언급한 그런 규칙(단일대입규칙이라 한다)이 만족된다는것을 강조해 둔다.

목표목록은 아래와 같이 정확히 계산된다.

Distance(1)=distance(temp2(2,1))=2,direction(1)=direction(temp2(2,1))=1
Distance(2)=distance(temp2(1,3))=1,direction(2)=direction(temp2(1,3))=3
Distance(3)=distance(temp2(1,4))=1,direction(3)=direction(temp2(1,4))=4
Distance(4)=distance(temp2(4,4))=4,direction(4)=direction(temp2(4,4))=4

이것을 푸는데서 중요한 문제는 Is Target함수가 목표의 분산이 어떻게 되였는가에 무관계하게 N*M번 평가되여야 한다.

HPF성능

이미 알고 있는 성능자료가 충분하지는 못하다 하더라도 정적병렬성과 정규통신패턴용 계산프로그람과 같은 두가지 측면을 놓고 Fortran 90과 HPF콤파일러의 성능을 몇가지 증거에 기초하여 평가해 보자. 이와 같은 의미에서 가장 일반적인것이라고는 볼수 없다.

실례로 표 14-3에서 제시된 자료를 살펴 보자. 이 자료는 HPF에 의한 쉘로우-워터(shallow-water)의 수값계산프로그람을 여러 형태의 콤퓨터에서 실행시켜 얻은 속도개선성능자료이다.

MPP들(Intel paragon과 IBM의 SP2)과 SMP (DEC의 AlphaS 8400) 그리고 DEC cluster에서 얻은 속도개선성능이 좋다는것을 알수 있었다.

어떤 프로쎄스에 대하여 3차Fortran 77프로그람과 HPF프로그람의 대응하는 실행결과의 비로 HPF의 속도개선정도를 평가하였다.

추가적인 최량화와 고속완충효과를 가지는것으로 하여 HPF콤파일러에 의한 결과가 훨씬 개선된것이라고 결론지울수 있다.

일부 응용에서 자료병렬성수법이 통보문넘기기수법에 대비할만한 성능을 가진다고 말할수 있다. 표 14-4에 꼭같은 알고리듬을 놓고 작성한 두가지 프로그람을 DEC Advantage Cluster에서 실현시킨 성능개선결과를 제시하였다. 이 두가지 프로그람가운데 하나는 Fortran 77과 PVM에 의한것이며 다른 하나는 HPF에 의한것이다.

표 14-3 HPF 코드의 성능개선

체계	처리기수			
	1	2	4	8
Intel Paragon	1.00	1.95	3.84	7.38
IBM SP2	1.00	1.97	3.81	7.50
DEC AlphaServer 8400	1.12	1.97	5.30	10.6
DEC Advantage Cluster	1.00	1.59	3.13	8.57

더우기 하나의 프로쎄스에서 실행되는 3차 Fortran 77코드와 비교되도록 속도개선값을 측정하였다.

보는바와 같이 자료병렬성코드는 모든 경우에 분명히 통보문넘기기코드를 휠씬 초월한다.

14. 4. 기타 자료병렬성수법

Fortran 90과 HPF의 보편성과 효과성

현재 부족점들을 줄이기 위한 일련의 연구가 진행되고 있다. 이 연구를 두가지 부류로 나누어 볼수 있는바 하나는 HPF와 류사한것이며 다른 하나는 Fortran이외의 언어에 기초한것이다.

우리는 자료병렬성를 목표로 하는 접속기계수법을 평가하는 형식으로 이 수법들을 아래에서 소개한다.

14. 4. 1. Fortran 95와 Fortran 2001

이제 **Fortran 95, Fortran 2001, Fortran D, Vienna Fortran**과 같은 4가지 대상에 대한 연구정형을 보자.

여기서 첫 2개는 Fortran 90을 확장한것이고 나머지 2개는 HPF를 확장한것이다.

이 책을 내놓을 때까지도 Fortran 90의 두가지 확장만이 국제규격화기구와 국제전자기술위원회（IEC）와 같은 국제규격화단체에 의하여 개발되고 있는 상태였다. 이 개발대상이 바로 Fortran 95와 Fortran 2001로 알려 져 있다.

1997년에 공개된 Fortran 95는 Fortran 90을 상대적으로 약간 개선한것이다. Fortran 95에서 새로운 주요특성은 FORALL명령문과 FORALL구성자, PORE와 ELEMENTAL 수속 및 구성체나 지적자, 디폴트초기화이다.

반면에 Fortran 2001은 2001년에 공개된것들가운데서 중요한 개발언어이다. Fortran 2001의 새로운 특성은 C언어와의 호상조작성, 유도자료형의 확장, 고성능수값프로쎄스를 위한 추가적지원, 비동기화된 I/O와 구동형 I/O와 같은 새로운 I/O특성, 객체지향의 고려, 조작체계를 통한 일련의 특징들이다.

확장된 HPF

Fortran D(Fortran 90D이후)와 ViennaFortran은 동적 및 비정규자료분포를 지원하는 보다 보편적인 자료넘기기기능들을 제공한다. 우리는 Fortran D의 기본사항을 실례를 들어 보기로 하자.

실례 14.9. HPF 코드로 푸는 N개체문제

N개의 개체문제에서 N개의 개체 X(1), X(2), …, X(n)체계의 동작은 아래와 같은 시간구간렬로 표현된다. 매 구간마다 매 개체 X(i)에 부과되는 force(i)는 다음과 같이 계산된다.

$$force(i) = \sum_{x(j) \text{ is a neighbor of } x(i)} f(i, j) \qquad (14.5)$$

($x(j)$ is a neighbor of $x(i)$: $x(j)$는 $x(i)$의 이웃)

여기서 개체 $x(j)$는 개체 $x(i)$와 그 사이거리가 어떤 턱값보다 작으면 서로 이웃이라고 말한다. 다음으로 매 개체의 위치와 속도는 새로 계산되는 force(i)에 의하여 갱신된다. 실례로 6개체 문제를 그림 14-6에 제시하였다.

그림에서 매 vertex는 개체를 표현한것이다.

물체의 쌍사이는 그것이 서로 이웃일 때 그어 진다. 턱값외에도 모두 6×5=30쌍으로 간주하여야 한다. 그림 14-6에서 우리는 우의 코드로 표현되는 7개 쌍을 고려하여야 한다.

조 A=(/1, 1, 2, 2, 3, 4, 3, …/)과 조 B=(/2, 5, 5, 3, 4, 6, 6, …/)의 경계 I의 2개의 수직선은 조 A(i), 조 B(i)로 표시된다. 례컨대 경계 4의 2개 사선은 조 A(4)=2및 조 B(4)=3이다.

ㄱ) 정규블로크분배

ㄴ) 비정규블로크분배

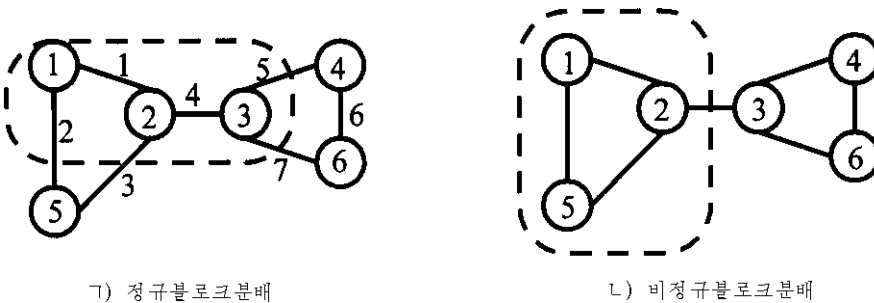



그림 14-6. 6개본체문제에 대한 자료분포

X(2)에 작용하는 힘은 그 세개이웃 X(1), X(3), X(5)로부터 받는다.

그러나 force(2)=f(2, 1)+f(2, 3)+f(2, 5)이다.

이것은 모든 이웃들로부터 유도된 합이다.

이 기능은 Fortran D의 FORALL에서 사용할수 있는 새로운 REDUCE interisic에 의하여 실현된다. 두개 마디에 대한 상세한 자료는 다음과 같이 서술된다.

```
!HPF$ PROCESSOR Nodes(2)
!HPF$ DISTRIBUTE x(BLOCK) ONTO Nodes
      B=(/1, 1, 2, 2, 1, 2/)
!HPF$ REDISTRIBUTE x(INDIRECT(B))
```

이 블로크분포는 그림 14-6 ㄱ)에 제시되였다.

점선구획안에 있는 세개의 개체는 마디 1에 배당되고 나머지 세개의 개체는 마디 2에 배당된다. 이 분포는 소유자계산규칙과 함께 통신되여야 할 6개의 부분을 산생시킨다.

하나는 간접적인 배렬 B를 통한 비정규분포에 의한 국부성을 개선할수 있다. 프로그람작성자는 처음에 간접배렬만을 지원한다. 다음에 동적 REDISTRIBUT명령문이 그림 14-6 ㄴ)에 제시된 형식으로 배렬 X의 가지들을 재배치한다.

정규분포에서 Block와 CYCLIC와는 달리 X(INDIRECTCB)는 X(i)가 Nodes(B(i))에 배치될것으로 본다. 그렇지만 X(s)는 Nodes(B(5)) 즉 Nodes(1)에 넘겨 져야 한다. 이 비정규분포는 통신되여야 할 두개의 가지만 요구한다.

HPF의 기본골격으로 서술한 이상의 모의를 요약하면 다음과 같다.

```
do time_step=1,number_of_time_steps
!compute the set of pairs
……
!compute the force on each particle
   FORALL(I=1:Number_of_Pairs)
        REDUCE(SUM,force(PairA(i)),f(PairA(i),PairB(i)))
        REDUCE(SUM,force(PairB(i)),f(PairA(i),PairB(i)))
   END FORALL

   FORALL(I=1:N) update_position_and_velocity(i)
end do          !시간단계되풀이의 끝
```

HPF를 완성하기 위한 개념들에 대한 연구를 HPF-2라고 한다.

HPF-2의 좋은 특성은 개성적인 응용프로그람들과 함께 문서화된 새 특성을 위한 합리성과 정당성이다. 그것들은 HPF에 어떤 확장된 능력이 추가되는가를 구체적으로 연시해 주는 실제적인 응용프로그람들이다. 우에서 론한 배렬의 비정규적넘기기 HPF-2연구에서 하나의 론점이 주소화되여 있다.

HPF-2를 위한 HPF연산에 의하여 고려되는 새로운 능력은 다음과 같다.

- 배렬만이 아닌 보다 보편적인 자료구조의 넘기기 그것은 련결목록이나 도출되는 자료형과 같은 지적자기초의 자료구조를 포함한다.
- 가상마디의 내부격자만이 아닌 처리기부분모임에로의 분포

- 소유자계산시간이 아닌 보다 유연한 작업부하배치도식
- 다중과제의(단일스레드만이 아닌) 동적생성과 완료들
- 원자적조작의 지원
- 병렬 I/O와 검사법
- 콤퓨터의 광대역스펙트르를 고성능실현을 위하여 설계된 HPF의 고전적부분모임으로서의 HPF핵심부

１４．４．２． pC++와 Nesl 방법

Fortran이 아닌 여러가지 언어를 쓰는 자료병렬성수법에는 여러가지가 있다. 실레로 인디아대학에서 개발한 pC++ 계획은 [98]객체지향수법을 적용하고 있다.

자료령역은 배렬만이 아닌 보다 유연한 대상에 의하여 정의된다. 그리고 불규칙자료병렬프로쎄스가 지원된다.

아래에서 CMU의 Blell 00h2에 의해 개발된 nesl언어의 일부를 상세히 론한다. pC++와 Nesl은 둘다 공통령역에서 실현된다. Nesl언어는 단순한 기초를 가진 자료병렬성언어에서 병렬알고리듬을 배우거나 정의하기 위한것이다. Nesl은 2가지 중요한 특성을 가진다. 그 하나는 자료요소의 순서모임을 지정하는 순서렬이다. 매 요소는 그 자체가 순서렬일수 있다. 정규자료구조로서의 배렬과는 달리 순서렬은 재귀를 포함한 규칙적 및 불규칙적자료구조를 다 표현할수 있다.

다른 하나는 그 자체가 순서렬일수 있는 모든 요소에 동시에 적용할수 있는 병렬함수를 가짐으로써 병렬성을 지원하는것이다.

이것은 평탄한（고르로운） 자료병렬성만을 지원하던 Fortran 90 또는 HPF와 비교된다.

Nesl의 주요병렬성구성체는 apply−to−each이다. 이 구성체는 식 ｛a*a:a in[1, 2, 3, 4]｜a>2｝에 의하여 직관적으로 설명할수 있다. 이 식은 "병렬로 순서렬 ｛1, 2, 3, 4｝에서 a>2인 매 요소 a에 두제곱연산을 적용한다"를 의미한다.

이것은 순서렬 [9,16]을 생성한다. 려파조인 a>2는 선택적이다.

Apply−to−each구성체에서 연산자는 연산자나 사용자정의함수가 될수 있다. 실레로 다음의 정의

Funcion faciorial(n)=**if**(n==1) **then** 1 **else** n*factorial(n−1);

에서 식 ｛factorial(b):bin[3, 2, 1]｝은 순서렬 [6, 2, 1]을 생성한다.

Nesl은 elementwise와 관계되는 모든 순서렬에 조작함수를 제공한다.

이와 같은 류의함수들을 표 14−4에 제시하였다.

보충적으로 **write**(s,p)함수는 2개의 변수를 가지는바 첫번째것은 순서렬 S이며 두번째것은 옹근수값조의 순서렬 P이다. 매 조（i,v）는 S의 I째 요소를 지정하며 v가 대입된다.

그러나 실행후에

Result=**write**([1,3,5,7],[(0,2),(3,3)])

이 된다.

우리는 Result=[2,3,5,3] 을 얻는다. 순서렬첨수가 0에서 시작된다는것을 강조해 둔다. 첨수를 P로 반복할수 있다. 이때 값들중 하나는 비결정론적방법으로 서술된다.

그러나

write([1,3,5,7],[(0,2),(0,4),(3,3)])

은 [2, 3, 5, 3]이나 [4, 3, 5, 3]을 귀환한다.

Nesl은 같은 병렬쓰기를 허용하지 않는 한쪽 배렬쓰기함수 **e_wirte**(S,P)를 제공한다.

표 14-4 여러가지 Nesl 순서렬함수

함 수	의 미	실 례
dist(*a*, *n*)	***a***의 ***n***개복사렬 복귀	**dist(t, 3) returns [t, t, t]**
#s	렬 **s**의 복쉬	**#[t,t,t] returns 3**
[*s*:*e*:*n*]	***n***에 의한 ***s***로부터 ***e***까지 옹근수렬복귀	**[2:9:3] returns [2, 5, 8]**
drop(*s*, *n*)	렬 ***s***의 첫번째 ***n***개원소 떨구기	**drop([1 ,2, 3, 4], 2) returns [3, 4]**
sum(*s*)	렬 ***s***의 감소	**sum([1, 2, 3]) returns 6**
flatten(*s*)	축소렬 ***s***를 안정	**flatten([[1,2],[[3],[4]]]) returns [1,2,3,4]**

실례 14.10. 성긴행렬의 동시합

우리는 성긴행렬의 매 행의 동시합을 얻으려 한다고 하자.

$$rowsum(A) = rowsum\left(\begin{bmatrix} 1.0 & 0 & 0 & 2.0 & 0 \\ 0 & 3.0 & 0 & 0 & 0 \\ 5.0 & 0 & 0 & 0 & 0 \\ 6.0 & 0 & 0 & 7.0 & 0 \end{bmatrix}\right) = \begin{bmatrix} 3.0 \\ 3.0 \\ 5.0 \\ 13.0 \end{bmatrix} = S \qquad (14.6)$$

대부분의 요소가 0이므로 이 행렬을 일부 특수한 형태로 기억기에 보존되도록 재표현할수 있다. 그 다음 병렬조작을 Fortran 90이나 HPF로 실현하는것은 어렵다. Nesl에서 성긴 행렬 A는 매개가（렬, 값）쌍으로 된 순서렬인 4개의 행의 순서렬로 표현할수 있다.

A=[[(0,1,0),(1,2,0)],[(1,3,0)],[(0,5,0)],[(0,6,0),(3,7,0)]]

이제 행의 합연산은 Nesl에서 간결하게 다음과 같이 지정될수 있다.

S={**sum**({v:(i,v)**in** row}):row **in** A}

이 실례는 수렬병렬성의 두 형태를 보여 주는바 자료순서렬 A는 축소화되여 있고 4개의 합계산은 병렬로 실행된다.

실례 14.11. 자료병렬성코드로 된 Evatosthenes 방법

N보다 작거나 같은 prim모두의 모임은 아래와 같은 Nesl수속을 리용하여 Eratosthenes 방법의 sieue에 의하여 찾을수 있다.

```
function primes(N)
if(N==2) then ([]int)
else
   let sqr_primes=primes(isqrt(N));
        composites={[2*p:N:p]:p in srq_primes};
        flat_comps=flatten(composites);
        flags=write(dist(t,N),{(i,f):i in flat_comps});
        indices={i in [0:N];f1 in flags | f1}
in drop(indices,2);
```

식 [] int는 N=2일 때 결과로 되는 빈순서렬을 표시한다. 함수 isqrt(N)는 N의 2차뿌리의 옹근수값을 귀환한다. 2개의 상수 t와 f는 론리값 true 혹은 false중 하나를 가리킨다.

N=20이라고 가정하자. primes(isqrt(20))=primes(4)를 찾을 때까지 재귀적으로 자기 자체로 접근하여 [2,3]을 귀환한다. 다른 변수들은 다음과 같은 값을 가진다.

Sqr_primes=[2,3]
Composites=[[4,6,8,10,12,14,16,18],[6,9,12,15,18]]
Flat_comps=[4,6,8,10,12,14,16,18,6,9,12,15,18]
Flags=[t,t,t,t,f,t,f,t,f,f,f,t,f,t,f,f,f,t,f,t]
Indices=[0,1,2,3,5,7,11,13,17,19]

그리하여 최종결과는 primes[20]에 의하여 귀환되는 [2, 3, 5, 7, 11, 17, 19]이다.

Blelloch는 두개의 측정량 즉 작업량과 깊이를 리용하여 병렬알고리듬의 성능을 평가할것을 주장하였다[87].

작업에 효과적인 병렬알고리듬(즉 여기서 병렬알고리듬은 가장 좋은 순차적알고리듬과 같은 량의 작업과 근사하게 수행한다.)에 대하여 작업량은 3장에서의 순차적인 작업부하 T_1과 상사적이며 깊이는 림계경로 T_∞와 류사하다.

그밖에 그 성능측정량들이 기계모형이거나 병렬프로그람선언에 기초하여야 한다고 제기하였다.

Nesl프로그람이 작업량과 깊이는 아래에 언급한바와 같이 견본연산(표 14−4에 제시되였다.)들과 측정량들을 표현식으로 결합하는 극치로부터 얻어 진다.

표 14-5　Nesl 견본함수들의 작업량과 깊이

연 산	작업	깊이	해 석
dist(*a,n*)	**1**	1	
#s	**1**	1	
[*s*:*e*:*n*]	**(e-s)/n**	1	
sum(s)	**L(s)**	**log(L(s))**	**L(s)**는 렬의 길이이다
drop(s,n)	**L(*result*)**	1	Result는 결과렬이다
flatten(s)	**L(result)**	**1**	
write(s,p)	**L(result)**	1	

표현식 e의 작업량과 깊이를 각각 $W(e)$, $D(e)$로 표시하자. 그러면 합성표현식의 두 측정렬들을 다음과 같은 부분표현식들로부터 계산할수 있다.

(1) 대다수 경우에 표현식의 작업량과 깊이를 그 부분표현식들의 작업량과 깊이값이다.
(2) 매개에 대하여 다음식을 적용한다.

$$W(\{e_1(a):\text{a in } e_2\}) = 1 + W(e_2) + \sum_{a \text{ in } e_2} W(e_1(a))$$
$$D(\{e_1(a):\text{a in } e_2\}) 1 + D(e_2) + \sum_{a \text{ in } e_2} D(e_1(a)) \qquad (14.7)$$

(3) "if의 then e_2 else e_3" 인 조건 e에 대하여 다음의 식을 리용한다.

$$¤W(e) = 1 + W(e_1) + \begin{cases} W(e_2) & e_1 = true \\ W(e_3) & e_1 = false \end{cases}$$
$$D(e) = 1 + D(e_1) + \begin{cases} D(e_2) & e_1 = true \\ D(e_3) & e_1 = false \end{cases} \qquad (14.8)$$

실례 14.12. 병렬알고리듬 primes(N)

실례 14.11에서 병렬알고리듬 primes(N)의 작업량과 길이는 무엇인가？평균병렬상은 얼마인가？

풀이

primes(N)이 f(N)의 작업량을 가진다고 하자.

Sqr_primes=primes(isqrt(N))에서 첫 재귀접근은 $f(\sqrt{N})$의 작업량을 가진다.

둘째 재귀접근은 $f(\sqrt[4]{N})$의 작업량을 가진다. 즉 문제의 크기는 문제의 크기가 2로 되는 마지막재귀준위에 도달될 때까지 N으로부터 $\sqrt{N}$, $\sqrt[4]{N}$으로 감소한다.

이 준위에서 총체적인 작업량은 $O(N\log\log N)$이며 임의의 재귀준위에서의 길이는 상수이다(문제 14.6볼것).

우리는 재귀준위를 가진다.

총 작업량은 $O(N\log\log N)$이며 총 깊이는 $O(\log\log N)$이며 평균병렬성은 $O(N\log\log N)/O(\log\log N)=O(N)$이다.

Nesl은 간단하고 잘 째여 져 있는 언어이다. 그것은 자료병렬, 함수의 형식, 잘 설계된 구성체들의 작은 모임을 리용하여 병렬계산알고리듬들을 간결하고 명확하게 규정할 수 있다는것을 론증한다. 이런 알고리듬들에서는 Nesl프로그람이 주류를 이루는 언어(병렬확정성을 가진 C언어와 Fortran)들로 규정된 대응하는 프로그람들보다 훨씬 더 간단하다. Nesl프로그람들은 형식에 의거하지 않는(혹은 간략적인) 알고리듬들에서 매우 가깝다.

이 모든 속성과 일반성은 Nesl이 자료병렬알고리듬을 교육하는 리상적인 언어로 되도록 한다. Nesl의 결합은 자료병렬과 함수형언어에서와 같다.그것은 MIND계산을 규정할만큼 주류를 이루지 못하며 유연하지 못한것이다. 실례로 은행경영문제와 목표탐색문제들을 푸는데서 Nes/로 병렬알고리듬을 서술하기 어렵다(12.13절을 볼것).

Nes에서의 또 다른 문제점은 무효시간을 줄이기 위한 자료위치항목과 병렬성통신을 충분히 고려하지 못한것이다. [89]는 DECstation 5000, CM−2, CM−5, Cray(90) 응용프로그람을 가지고 Nesl성능실험을 진행하였다. Nesl프로그람의 성능은 정밀도자료에 대하여서는 기계전용인 Fortran 또는 C코드와 견줄만 하다는것을 보여 준다. 흔히 비정규자료에 대하여서도 우월하다.

14. 5. 참고문헌주해와 련습문제

자료병렬수법은 [310], [20], [89], [142]들에서 론의했다. [142]는 자료병렬분야에 대하여 총괄적으로 론의하였다. Fortran 90은 [4]에서, Fortran 95는 [5]에서 론의되였다. 고성능 Fortran은 [370]과 [419]에서 론의되였다. HPF의 확장과 그의 특성은 [149]와 [506, 507]에서 론의하였다.

실례 14.9는 [507]에 따른것이다.

자료병렬성과 Nesl언어는 [87, 89]에서 론의되였다.

이 장에서 Nesl에 대한 론의들은 우의 두 론문에서 최대로 취한자료에 기초한것이다.

사유하는 기계연구집단은 C*, Fortran, CMFortran, *Lisp와 같은 여러가지 유력한 자료병렬언어들을 개발하였다.

문 제

문제 14.1. 그림 14−1에서 야꼬비완화코드를 리용하여 다음의 자료병렬프로그람의 특징들을 설명하시오.

(1) 조종단일스레드
(2) 전역적이름붙이기공간
(3) 집합적자료구조우에서의 병렬연산
(4) 약한 동기화
(5) 암시적호상작용
(6) 암시적자료배정

문제 14.2. 대각선화단계에서의 작업량이 그림 14−5보다 더 균형화되는 HPM에서 가우스소거프로그람을 작성하시오.

(1) 렬크기 N과 기계크기 *n*에 대한 프로그람을 작성하시오.
(2) N이 *n*보다 훨씬 더 크며 N이 건으로 고르롭게 나누어 진다는것들의 가정을 간단히 할수 있다.
(3) 두 마디체계를 가정하자. (1)의 프로그람에 대하여 두 마디중 매 마디에 대한 flop작업량이 얼마인가?
(4) 두 마디체계를 가정하자.그림 14−5의 프로그람인 경우에 두마디중 매 마디에 대한 flop작업량은 얼마인가?

문제 14.3. 야꼬비완화를 위한 HPF프로그람을 작성하시오(실례 12.4와 그림 14−18을 보시오).

HPF코드가 그림 14−1보다 얼마나 더 단순해지며 고속화되는가를 설명하시오.

주의 : HDF코드는 배렬 Diag A를 계산하는데 순차적인 do순환고리를 요구하지 않는다.

문제 14.4. 자료병렬성에 대한 Fortran 90과 HPF방법을 비교하고 다음 문제에 대답하시오.

(1) Fortran 90과 HPF방법에서 intrinsic함수에서의 류사성과 차이점은 무엇인가?
(2) 병렬배렬연산에 대하여 (1)부분과 같은 문제를 반복하시오.
(3) 100p−Carried를 조종하는데서 그것이 HDF에서는 어떻게 지원되는가를 설명하시오.
(4) Fortran 90과 HDF에서 자료병렬성에 대한 콤파일러지원의 우결함을 비교하시오.

오늘의 Fortran 90과 HDF콤파일러들의 한계는 무엇인가?

문제 14.5. Eratostenes방법의 선별에 의하여 N과 같거나 더 작은 모든 primes들의 모임을 찾는 프로그람을 작성하시오(실례 14.11).

(1) Fortran 90으로
(2) HPF로
(3) 실례 14.12를 가지고 이 프로그람을 비교하여 Fortran 90, HPF Nesl의 우결함을 분석하시오.

문제 14.6. 실례 14.11에서 prime알고리듬이 재기준위 0에서 0(N1og1ogN)만한 작업량과 0(1)만한 깊이를 가지는가를 확인하시오.

문제 14.7. Fortran 90과 HPF개선을 위한 연구결과들을 고려하여 이 장에서 배운것외의 가능 및 특징확장에 대한 최신개발을 더 깊이 고찰하여 다음 질문에 대답하시오.
(1) 오늘의 Fortran 90으로부터 그 특성상 Fortran 95보다 개선된것이 무엇인가?
(2) 포트란 2001에 대하여 부분(1)을 반복하시오.

문제 14.8. 자료병렬성을 리용하는 이 Fortran수법들을 고려해서 다음 질문에 대답하시오.
(1) Indiana종합대학의 PC++과제를 연구하여 자료병렬성에 대한 객체지향적방법에 주해를 붙이시오.
(2) Neal언어로 개발된 순서렬의 새로운 착상을 론하시오.
(3) 응용프로그람들에서 자료병렬성을 조종하는 객체지향적이며 함수적인 방법들을 리용하는데서의 약점을 밝히시오.

문제 14.9. 명시적병렬성에 대한 세가지 병렬프로그람작성모형을 모두 배웠다. 표 12.3을 다시 보고 다음의것에 대한 완전한 주장을 가지고 대답하시오.
(1) 병렬성조종, 통신, 동기화, 유한성, 확정성, 수정성, 복잡성, 이식성과 같은 자료병렬방법이 가장 리득을 보는 항목들에 관하여 세가지 모형들의 상대적순위를 비교판단하시오.
(2) 비정극성조종, 응용프로그람의 일반성, 효과성과 같이 자료방법 모형의 순위를 정하는데서 가장 기약하다고 보는 내용에 관하여 부문(1, 2)를 반복하시오.
(3) 집합성과 일반성, 효과성에 관하여 통보문넘기기모형에 대하여 (1)를 반복하시오.
(4) 병렬성, 배치, 통신, 구성체계에 관하여 통보문넘기기모형을 놓고 (2)를 반복하시오.
(5) 배치, 통신, 비정규성, 일반성에 관하여 공유기억모형을 놓고 (1)를 반복하시오.
(6) 동기화, 집합성, 4가지 의미론적문제점들, 프로그람이식성에 관하여 공유기억모형을 놓고 (2)를 반복하시오.

문제 14.10. 표 12.3의 자료들은 프로그람작성능력에 관하여 일반성을 제외한 모든 문제들에서 가장 좋은 비률이라는것을 보여 준다. 여기서 독자의 주장이 문제 12.2에서 주어 진것보다 더 심오하여야 한다. 이 요구를 례증하기 위하여 가동환경의존실례를 리용하면서 성능평가실험결과를 인용하시오.

참 고 문 헌

[1] B. Abeli, C. B. Stunkel, and C. Benveniste, "Clock Synchronization on a Multicomputer," *Journal of Parallel and Distributed Computing*, Vol. 40, No. 1, 1997, pp. 118-130.

[2] ACM, *Resources in Parallel and Concurrent Systems*, with an Introduction by Charles Seitz, ACM Press, New York, 1991.

[3] D. Adams, *Cray T3D System Architecture Overview Manual*, Cray Research, Inc., Sept. 1993.

[4] J. Adams, W. Brainerd, J. Martin, B. Smith, and J. Wagener, *The Fortran 90 Handbook*, McGraw-Hill, New York, 1992.

[5] J. Adams, W. Brainerd, J. Martin, B. Smith, and J. Wagener, *The Fortran 95 Handbook*, MIT Press, 1997.

[6] S. V. Adve and M. D. Hill, "A Unified Formalization of Four Shared-Memory Models," *IEEE Trans. on Parallel and Distributed Systems*, June 1993, Vol. 4, No. 6, pp. 13-24.

[7] S. V. Adve and M. D. Hill, "Weak Ordering: A New Definition," *Proc. 17th Ann. Int'l. Symp. Computer Arch.*, 1990.

[8] S. V. Adve and K. Gharachorloo, "Shared Memory Consistency Models: A Tutorial," *IEEE Computer*, Vol. 29, No. 12, Dec. 1996, pp. 66-76.

[9] A. Agarwal, "Performance Tradeoffs in Multithreaded Processors," *IEEE Trans. Parallel and Distributed Systems*, May 1992, pp. 525-539.

[10] A. Agarwal, R. Bianchini, D. Chaiken, K.L. Johnson, D. Krantz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie, and D. Yeung, "The MIT Alewife Machine: Architecture and Performance," *Proc. of the 22nd Annual International Symposium on Computer Architecture*, pp. 2-13, 1995.

[11] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An Evaluation of Directory Schemes for Cache Coherence," *Proc. 15th Int'l. Symp. on Computer Architectue*, pp. 280-289, June 1988.

[12] R. C. Agarwal et al., "High-Performance Implementations of the NAS Kernel Benchmarks on the IBM SP2," *IBM System Journal*, Vol. 34, No. 2, 1995, pp. 263-272.

[13] R. C. Agarwal, F. G. Gustavson, and M. Zubair, "Exploiting Functional Parallelism of POWER2 to Design High-Performance Numerical Algorithms," *IBM J. Res. Develop.*, Vol. 38, No. 5, 1994, pp. 563-576.

[14] T. Agerwala, J. L. Martin, J. H. Mirza, D. C. Sadler, D. M. Dias, and M. Snir, "SP2 System Architecture," *IBM Systems Journal*, Vol. 34, No. 2, 1995, pp. 152-184.

[15] G. Agha, "Concurrent Object-Oriented Programming," *Comm. of the ACM*, Vol. 33, No.9, Sept, 1990, pp. 125-141.

[16] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[17] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, Reading, Mass., 1988.

[18] M. Ajtai, J. Komlos, and E. Szemeredi, "An $O(n\log n)$ Sorting Network," *Proc. 15th ACM Symp. on Theory of Computing*, 1983, pp. 1-9.

[19] R. Alasdair, A. Bruce, J. G. Mills, and A. G. Smith; "CHIMP Version 2.0 User Guide," University of Edinburgh, March 1994.

[20] E. Albert, J. Lukas, and G. Steele, "Data Parallel Computers and the FORALL Statement" *J. Parallel and Distributed Computing*, 13(2):185-192, 1991.

[21] T. B. Alexander et al., "Corporate Business Servers: An Alternative to Mainfrmaes for Business Computing," *Hewlett-Packard Journal*, June 1994, pp. 8-33.

[22] A. Alexandrov, M. Ionescu, K. Schauser, and C. Scheiman, "LogGP: Incorporating Long Messages into the LogP Model," *Proc. of the 7th Annual ACM Symp. on Parallel Algorithms and Architectures*, 1995, pp. 95-105.

[23] M. S. Allan and M. C. Becker, "Multiprocessing Aspects of the PowerPC 601," *COMPCON*, Spring 1993.

[24] S. J. Allan and R. Oldehoeft, "HEP SISAL: Parallel Functional Programming", in Kowalik (Ed.), *Parallel MIMD Computation: HEP Supercomputers and Applications*, MIT Press, Cambridge, Mass., 1985.

[25] V. H. Allan, R. B. Jones, R. M. Lee, S. J. Allan, "Software Pipelining," *ACM Computing Survey*, 27(3):367-432, 1995.

[26] G. S. Almasi and A. J. Gottlieb, *Highly Parallel Computing*, 2d ed., Benjamin/Cummings, New York, 1993.

[27] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith, "The Tera Computer System," in *1990 Int'l. Conf. on Supercomputing*, pp. 1-6, September 1990.

[28] S. P. Amarasingbe, J. M. Anderson, C. S. Wilson, S.-W. Liao, B. R. Murphy, R. S. French, M. S. Lam, and M. W. Hall, "Multiprocessors from a Software Perspective," *IEEE Micro*, Vol. 16, No. 3, 1996, pp. 52-61.

[29] G. M. Amdahl, "Validity of Single-Processor Approach to Achieving Large-Scale Computing Capability," *Proc. AFIPS Conf.*, Reston, Va., 1967, pp. 483-485.

[30] C. Amza, A.L. Cox, S.D. Warkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, "TreadMarks: Shared Memory Computing on Networks of Workstations," *IEEE Computer*, 29(2):18-28, 1996.

[31] D. Anderson and T. Shanley, *Pentium Processor System Architecture*, 2d ed., Addison-Wesley, Reading, Mass., 1995.

[32] J. Anderson and M. Moir, "Using Local-Spin *k*-Exclusion Algorithms to Improve Wait-Free Object Implementations ," *Distributed Computing*, to appear 1997.

[33] T.E. Anderson, D.E. Culler, D.A. Patterson et al., "A Case for NOW (Networks of Workstations)," *IEEE Micro*, February 1995, pp. 54-64.

[34] T.E. Anderson, M.D. Dahlin, J.M. Neefe, D.A. Patterson, D.S. Roseli, and R.Y. Wang, "Serverless Network File Systems," *ACM Transactions on Computer Systems*, Vol. 14, No. 1, 1996, pp. 41-79.

[35] J. Anderson and M. Lam, "Global optimizations for Parallelism and Locality on Scalable Parallel Machines," *Proc. of the SIGPLAN'93 Conf. on Programming Language Design and Implementation*, 1993.

[36] T. E. Anderson, H. M. Levy, B. N. Bershad, and E .D. Lazowska, "The Interaction of Architecture and Operating System Design," *Proc. 4th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1991, pp. 108-120.

[37] G.R. Andrews, *Concurrent Programming: Principles and Practice*, Benjaming/Cummings, Redwood City, California, 1991.

[38] I. Angus, G. Fox, J. Kim, and D. Walker, *Solving Problems on Concurrent Processors*, Vol. II, Prentice-Hall, Englewood Cliffs, N.J., 1990.

[39] ANSI Technical Committee X3H5, *Parallel Processing Model for High-Level Programming Languages*, 1993.

[40] R. H. Arpaci, D.E. Culler, A. Krishnamurthy, S.G. Steinberg, and K. Yelick, "Empirical Evaluation of the Cray T3D: A Compiler Perspective," *Proc. the 22d Annual Int'l. Symp. on Computer Architecture*, 1995, pp. 320-331.

[41] K.R. Apt and E.-R. Olderog, *Verification of Sequential and Concurrent Programs*, Springer-Verlag, New York, 1991.

[42] J. Archibald and J.L. Baer, "Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," *ACM Trans. Computer Systems*, Vol. 4, March 1986, pp. 273-296.

[43] B.W. Arden and H. Lee, "Analysis of Chordal Ring Network," *IEEE Trans. on Computers*, Vol. 30, No.4, April 1981, pp. 291-295.

[44] R. H. Arpaci, A. C. Dusseau, A. M. Vahdat, L. T. Liu, T. E. Anderson, and D. A. Patterson, "The Interaction of Parallel and Sequential Workloads on a Network of Workstations," *Proc. of the 1995 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, 1995, pp. 267-278.

[45] W. C. Athas and C. L. Seitz, "Multicomputers: Message-Passing Concurrent Computers," *IEEE Computer,* August 1988, pp. 9-24.

[46] ATM Forum, *ATM User Network Interface: Version UNI 3.1 Specification*, March 1994.

[47] A. Azagury, D. Dolev, G. Goft, J. Marberg, and J. Satran, "Highly Available Cluster: A Case Study," *Proc. 24th Symp. on Fault-Tolerant Computing*, 1994, pp. 404-413.

[48] R.G. Babb II (Ed.), *Programming Parallel Processors*, Addison-Wesley, Reading, Mass., 1988.

[49] M. J. Bach and S. J. Buroff, "Multiporcessor UNIX Operating Systems", *AT&T Bell Lab. Tech. Journal*, 63(8), Oct. 1984.

[50] J. Bacon, *Concurrent Systems: An Integrated Approach to Operating Systems, Database, and Distributed Systems*, Addison-Wesley, Reading, Mass., 1993.

[51] D. A. Bader, D. R. Helman, and J. JaJa, "Practical Parallel Algorithms for Personalized Communication and Integer Sorting," *UMIACS Technical Report*, University of Maryland, Dec. 1995.

[52] J. L. Baer and W .H. Wang, "Multi-Level Cache Hierarchies: Organizations, Protocols, and Performance," *J. of Parallel and Distributed Computing,* Vol. 6, 1989, pp. 451-476.

[53] J.C.M. Baeten and W.P. Weijland, *Process Algebra*, Cambridge University Press, 1990.

[54] D.H. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow, "The NAS Parallel Benchmarks 2.0," NASA Ames Research Center Report NAS-95-020, Dec. 1995.

[55] W. E. Baker, R. W. Horst, D. P. Sonnier, and W. J. Watson, "A Flexible ServerNet-Based Fault-Tolerant Architecture," *25th Annual Symp. on Fault-Tolerant Computing*, June 1995.

[56] M. A. Baker, G. C. Fox and H. W. Yau, "Cluster Computing Review," Northeast Parallel Architectures Center, Syracuse University, Nov. 1995.

[57] H. E. Bal, M.F. Kaashoek, and A.S. Tanenbaum, "Orca: A Language for Parallel Programming of Distributed Systems," *IEEE Trans. Soft. Eng.*, March 1992, pp. 190-205.

[58] H. E. Bal, J. G. Steiner, and A. S. Tanenbaum, "Programming Languages for Distributed Computing Systems," *ACM Computing Surveys*, 21(3):261-322, 1989.

[59] V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C.-T. Ho, S. Kipnis, and M. Snir, "CCL: A Portable and Turnable Collective Communication Library for Scalable Parallel Computers," *IEEE Trans. on Parallel and Distributed Systems*, 1995, 6(2):154-164.

[60] U. Banerjee, *Dependence Analysis*, Kluwer Academic Publishers, Boston, 1996.

[61] J. Barlett et al., "Fault Tolerance in Tandem Computer Systems," in *Reliable Compute Systems: Design and Evaluation*, D. Siewiorek and R. Swarz(Eds.), Digital Press, Maynard, Mass., 1992.

[62] R. Baron et al., "Mach-1: An Operating Environment for Large-Scale Multiprocessor Applications," *IEEE Software*, July 1985, pp. 65-67.

[63] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods,* SIAM Press, Philadelphia, 1992.

[64] B. E. Bauer, *Practical Parallel Programming*, Academic Press, San Diego, Calif., 1992.

[65] BBN, *TC2000 Technical Product Summary,* BBN Advanced Computers Inc., Cambridge, Mass., Nov. 1989.

[66] A. Beguilin, J. Dongarra, G. A. Geist, R. Manchek, and V. S. Sunderam, *A Users' Guide to PVM,* Technical Report ORNL TM-11286, Oak Ridge National Lab., July 1991.

[67] G. Bell, "Ultracomputers: A Teraplop before Its Time," *Communications of the ACM*, August 1992, pp. 27-47.

[68] G. Bell, "An Insider's Views on the Technology and Evolution of Parallel Computing," in *Software for Parallel Computers*, R. H. Perrott (Ed), Chapman & Hall, London, 1992, pp 11-26.

[69] G. Bell, "Why There Won't Be Apps: The Problem with MPPs," *IEEE Parallel and Distributed Technology,* Fall 1994, pp. 5-6.

[70] G. Bell, "1995 Observations on Supercomputing Alternatives: Did the MPP Bandwagon Lead to a Cul-de-Sac?" *Communications of the ACM.*, Vol. 39, No. 3, 1996, pp. 11-15.

[71] G. Bell and J. N. Gray, "The Revolution Yet to Happen," in P.J. Denning and R.M. Metcalfe (Eds.), *Beyond Calculation: The Next Fifty Years of Computing*, Springer-Verlag New York, Inc., 1997.

[72] M. Ben-Ari, *Principles of Concurrent and Distributed Programming*, Prentice-Hall, Englewood Cliffs, N.J., 1990.

[73] R. J. Bergeron, "The Performance of the NAS HSPs in 1st Half of 1994," Report NAS-95-008, NASA Ames Research Center, February 1995.

[74] A. J. Bernstein, "Analysis of Programs for Parallel Processing," *IEEE Trans. Elect. Comput.*, Vol. EC-15, No. 5, 1966, pp. 757-763.

[75] P.A. Bernstein and E. Newcomer, *Principles of Transaction Processing*, Morgan Kaufmann Publishers, San Francisco, 1997.

[76] M. Berry et al. "The Perfect Club Benchmarks: Effective Performance Evaluation of Supercomputers". *Technical Report* CSRD No. 827, Center for Supercomputing Research and Development, University of Illinois, Urbana, Ill., May 1989.

[77] J. Berstin, "Sybase for HACMP/6000: An Architectured Approach to Clustered Systems," IBM *AIXpert*, May 1994, pp. 46-52.

[78] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, New Jersey, 1989.

[79] P. Bhat, Y. Lim, and V. Prasanna, "Scalable Portable Parallel Algorithms for STAP," *Proc. of the Adaptive Sensor Array Processing Workshop*, MIT Lincoln Lab., March 1996.

[80] L. N. Bhuyan, R. Iyer, T. Askar, A. Nanda, and M. Kumar, "Performance of Multistage Bus Networks for a Distributed Shared Memory Multiprocessor," *IEEE Trans. on Parallel and Distributed System*, Jan. 1997, pp. 82-95.

[81] L. N. Bhuyan and X. Zhang, *Multiprocessor Performance Measurement and Evaluation*, IEEE Computer Society Press, 1995.

[82] A. Bilas and E.W. Felten, "Fast RPC on the SHRIMP Virtual Memory Mapped Network Interface," *Journal of Parallel and Distributed Computing*, Vol. 40, No. 1, 1997, pp. 138-146.

[83] R. Bisiani and M. Ravishankar, "PLUS: A Distributed Shared-Memory System," *Proc. 17th Int'l. Symp. on Computer Architecture*, May 1990, pp. 115-124.

[84] P. Bitar and A. M. Despain, "Multiprocessor Cache Synchronization Issues, Innovations, and Evolution," *Proc. 13th Annu. Int'l. Symp. Computer Arch.*, 1986.

[85] D. L. Black, "Scheduling Support for Concurrency and Parallelism in the Mach Operating System," *IEEE Computer*, Vol. 23, No. 5, May 1990, pp. 35-42.

[86] G. E. Blelloch, "Scans as Primitive Parallel Operations and Their Use in Algorithm Design," *IEEE Trans. on Computers*, TC-38(11), 1989, pp. 1526-1538.

[87] G. E. Blelloch, "Programming Parallel Algorithms," *Comm. of ACM*, Vol. 39, 1996, pp. 85-97.

[88] G. E. Blelloch, K.M. Chandy, and S. Jagannathan (Eds.), *Specification of Parallel Algorithms*, American Mathematical Society, 1994.

[89] G. E. Blelloch, S. Chatterjee, J.C. Hardwick, J. Sipelstein, and M. Zagha, "Implementation of a Portable Nested Data Parallel Language," *Journal of Parallel and Distributed Computing*, Vol. 21, No. 1, 1994, pp. 4-14.

[90] D. W. Blevins, E. W. Davies, R. Heaton, and J. H. Reif, "BLITZEN: A Highly Integrated Massively Parallel Machine," *Journal Parallel and Distributed Computing*, 1990, pp. 150-160.

[91] W. Blume and R. Eigenmann, "Performance Analysis of Parallelizing Compilers on the Perfect Benchmarks Programs," *IEEE Trans. on Parallel and Distributed Systems*, 1992, Vol. 3, No. 6, pp. 643-656.

[92] W. Blume, R. Eigenmann, J. Hoeflinger, D. Padua, P. Petersen, L. Rauchwerger, and P. Tu, "Automatic Detection of Parallelism: A Grand Challenge for High-Performance Computing," *IEEE Parallel and Distributed Technology*, 2(3):37-47, 1994.

[93] W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, T. Lawrence, J. Lee, D. Padua, Y. Paek, B. Pottenger, L. Rauchwerger, and P. Tu, "Parallel Programming with Polaris," *IEEE Computer*, Vol. 29, No. 12, Dec. 1996, pp. 78-82.

[94] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: An Efficient Multithreaded Runtime System," *Journal of Parallel and Distributed Computing*, Vol. 37, No. 1, 1996, pp. 55-69.

[95] M. A. Blumrich, C. Dubnicki, E.W. Felten, and K. Li, "Protected, User-Level DMA for the SHRIMP Network Interface," *Proc. 2nd Int'l. Symp. on High-Performance Computer Architecture*, Feb. 1996.

[96] M. A. Blumrich, C. Dubnicki, E.W. Felten, K. Li, and M.R. Mesarina, "Virtual-Memory-Mapped Network Interfaces," *IEEE Micro*, Feb. 1995, pp. 21-27.

[97] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro*, Feb. 1995, pp. 29-36.

[98] F. Bodin, P. Beckman, D. B. Gannon, S. Narayana, and S. Yang, "Distributed pC++: Basic Ideas for an Object Parallel Language," *Proc. Supercomputing '91*, pages 273-282, 1991.

[99] S. H. Bokhari, "Multiphase Complete Exchange: A Theoretical Analysis," *IEEE Trans. on Computers*, Vol. 45, No. 2, 1996, pp. 220-229.

[100] S. H. Bokhari, "Multiphase Complete Exchange on Paragon, SP2, annd CS-2," *IEEE Parallel and Distributed Technology*, Vol. 4, No. 3, 1996, pp. 45-59.

[101] R. Bond, "Measuring Performance and Scalability Using Extended Versions of the STAP Processor Benchmarks," *Technical Report*, MIT Lincoln Laboratories, Lexington, MA. December 1994.

[102] B. Boothe and A. Ranade, "Improved Multithreading Techniques for Hiding Communication Latency in Multiprocessor," *Proc. 19th Annu. Int'l. Symp. Computer Arch.*, Australia, May 1992.

[103] R. Bordawekar, J. del Rosario, and A. Choudhary, "Design and Evaluation of Primitives for Parallel I/O," *Proc. Supercomputing '93*, pp. 452-461, 1993.

[104] R. Bordawekar, A. Choudhary, K. Kennedy, C. Koelbel, and M. Paleczny, "A Model and Compilation Strategy for Out-of-Core Data Parallel Programs," *Proc. of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, August 1995, pp. 1-17.

[105] W. J. Bouknight, S. A. Denenberg, D. E. McIntyre, J. M. Randall, A. H. Sameh, and D. L. Slotnick, "The Illiac IV System," *Proceedings of the IEEE*, Vol. 60, No. 4, April 1972, pp. 369-388.

[106] N. S. Bowen, C. A. Polyzois, and R. D. Regan, "Restart Services for Highly Available Systems," *Proc. 7th IEEE Symp. on Parallel and Distributed Processing*, Oct. 1995, pp. 596-601.

[107] S. Brawer, *Introduction to Parallel Programming*, Academic Press, New York, 1989.

[108] M. Brehob, T. Doom, R. Enbody, W.H. Moore, S. Q. Moore, R. Sass, and C. Severance, "Beyond RISC - The Post-RISC Architecture," *Technical Report CPS-96-11*, Dept. of Computer Science, Michigan State University, 1996.

[109] R. P. Brent, "The Parallel Evaluation of General Arithmetic Expressions," *Journal of the ACM*, Vol. 21, No. 2, 1972, pp. 201-206.

[110] T. Brewer and G. Astfalk, "The Evolution of the HP/Convex Examplar," *Proc. of COMPCON Spring '97*, IEEE Computer Society, Feb.1997, pp. 81-86.

[111] P. Brinch Hansen, *Studies in Computational Science: Parallel Programming Paradigms*, Prentice-Hall, Englewood Cliffs, N.J., 1995.

[112] J. Bruck, D. Dolev, C.-H. Ho, M.C. Rosu, and R. Strong, "Efficient Message Passing Interface (MPI) for Parallel Computing on Clusters of Workstations," *Journal of Parallel and Distributed Computing*, Vol. 40, No. 1, 1997, pp. 19-34.

[113] J. Bruck, L. de Coster, N. Dewulf, C.-T. Ho, and R. Lauwereins, "On the Design and Implementation of Broadcast and Global Combine Operations Using the Postal Model," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, No. 3, March 1996, pp. 256-265.

[114] H. Bryhni and B. Wu, "Initial Studies of SCI LAN Topologies for Local Area Clustering," the *First International Worksthop on SCI-Based Low-Cost/High-Performance Computing*, Santa Clara University, August 1994.

[115] D. Burger and J.R. Goodman, "Guest Editors' Introduction: Billion-Transistor Architectures," *IEEE Computer Magazine*, Vol.30, No. 9, Sept. 1997, pp. 46-48.

[116] H. Burkhart III et al., *Overview of the KSR1 Computer System*, Technical Report KSP-TR-9202001, Kendall Square Research, Boston, February 1992.

[117] G. Burns, R. Daoud, and J. Vaigl, "LAM: An Open Cluster Environment for MPI," Ohio Supercomputer Center, (http://www.osc.edu/lam.html), May 1994.

[118] R. Butler and E. Lusk, "Monitors, Message, and Clusters: The p4 Parallel Programming System," *Parallel Computing*, Vol. 20, April 1994, pp. 547-564.

[119] G. Cabillic and I. Puaut, "Stardust: An Environment for Parallel Programming on Networks of Heterogeneous Workstations," *J. Parallel and Distributed Computing*, Vol. 40, No. 1, 1997, pp. 65-80.

[120] B. Callaghan, "WebNFS: The Filesystem for the World Wide Web," *SunSoft White Paper*, 1996.

[121] D. Callahan, K. Cooper, R. Hood, K. Kennedy, and L. Torczon, "ParaScope: A Parallel Programmign Environment", *Int. Journal of Supercomputer Applications*, 2(4), 1988.

[122] K. Cameron, L. J. Clarke, and A. G. Smith, "CRI/EPCC MPI for Cray T3D," http://www.epcc.ed.ac.uk/t3dmpi/Product/, August 1996.

[123] D. C. Cann, J. T. Feo, and T. M. DeBoni, "SISAL 1.2: High-Performance Applicative Computing," *Proc. Symp. Parallel and Distributed Processing*, 1990, pp. 612-616.

[124] P. Cao, E. W. Felten, A. R. Karlin, and K. Li, "A Study of Integrated Prefetching and Caching Strategies," *Proceedings of the 1995 ACM SIGMETRICS*, 1995.

[125] J. Cao, W. Jia, X. Jia, and T.Y. Cheung, "Design and Analysis of an Efficient Algorithm for Coordinated Checkpointing in Distributed Systems," *Proc. of 1997 IEEE Symp. Reliable Distributed Systems*, March 1997, pp. 261-268.

[126] W. M. Cardoza, F. S. Glover, and W. E. Snaman, "Overview of Digital UNIX Cluster System Architecture," *Proc. of COMPCON'96*, Feb. 1996, pp. 254-259.

[127] W. M. Cardoza, F.S. Glover, W.E. Snaman, Jr., "Design of the TruCluster Multicomputer System for the Digital Unix Environment," *Digital Technical Journal*, Vol. 8, No. 1, 1996, pp. 5-17.

[128] G. F. Carey (Ed.), *Parallel Supercomputing: Methods, Algorithms and Applications*, Wiley, New York, 1989.

[129] N. Carriero and D. Gelernter, *How to Write Parallel Programs*, MIT Press, Cambridge, Mass., 1990.

[130] N. Carriero and D. Gelernter, "Linda in Context," *Comm. ACM*, 32(4):444-458, 1989.

[131] J. B. Carter, J. K. Bennett, and W. Zwaenepoel, "Techniques for Reducing Consistency-Related Communication in Distributed Shared Memory Systems," *ACM Trans. on Computer Systems*, 13(3) :205-243, 1995.

[132] B. Catenzaro, *Multiprocessor System Architectures: Multithreaded Systems Using SPARC, Multilevel Bus Architectures, and Solaris (SunOS)*, SunSoft Press, Mountain View, Calif., 1994.

[133] C. Catlett and L. Smarr, "Metacomputing," *Comm. of ACM*, Vol. 35, No. 6, June 1992, pp. 44-52.

[134] M. Cekleov et al., "SPARCcenter 2000: Multiprocessing for the 90's," *Digest of Papers. COMPCON Spring '93*, San Francisco, IEEE Comput. Soc. Press, 1993, pp. 345-53.

[135] L. Censier and P. Feautrier, "A New Solution to Coherence Problems in Multicache Systems," *IEEE Trans. on Computers*, C-27(12): 1112-1118, December 1978.

[136] Y. Censor and S.A. Zenios, *Parallel Optimization: Theory, Algorithms, and Applications*, Oxford University Press, New York, 1997.

[137] D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal, "Directory-Based Cache Coherence in Large-Scale Multiprocessors", *IEEE Computer* 23(6): 49-58, June 1990.

[138] R. Chandra, S. Devine, B. Verghese, A. Gupta, and M. Rosenblum, "Scheduling and Page Migration for Multiprocessor Compute Servers," *Proc. 6th Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 12-24, 1994.

[139] R. Chandra, A. Gupta, and J. Hennessy, "COOL: An Object-Based Language for Parallel Programming," *Computer*, 27(8):14-26, 1994.

[140] S. Chandra, J. R. Larus, and A. Rogers, "Where Is Time Spent in Message-Passing and Shared-Memmory Programs?" *Proc. 6th Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 61-73, 1994.

[141] K. M. Chandy and I. Foster, "A Deterministic Notation for Cooperating Processes," *IEEE Trans. Parallel and Distributed Systems*, 1995.

[142] K. M. Chandy, I. Foster, K. Kennedy, C. Koelbel, and C.-W. Tseng, "Integrated Support for Task and Data Parallelism," *Int'l. J. Supercomputer Applications*, 8(2): 80-98, 1994.

[143] K. M. Chandy and C. Kesselman, "Parallel Programming in 2001," *IEEE Software*, 1991, 8(6):1-20.

[144] K. M. Chandy and C. Kesselman, "CC++: A Declarative Concurrent Object-Oriented Programming Notation," *Research Directions in Concurrent Object-Oriented Programming*, MIT Press, 1993.

[145] K. M. Chandy and S. Taylor, *An Introduction to Parallel Programming*, Jones and Bartlett, 1992.

[146] A. Chang and M. F. Mergen, "801 Storage: Architecture and Programming," *ACM Trans. Computer Systems*, 6(1): 28-50, 1988.

[147] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W.W. Hwu, "IMPAXR: An Architectural Framework for Multiple-Instruction Issue Processors," *Proc. of Int'l. Symp. Computer Architecture*, 1991, pp. 226-232.

[148] J. Chapin, S.A. Herrod, M. Rosenblum, and A. Gupta, "Memory System Performance of Unix on CC-NUMA Multiprocessors," *Proc. of the 1995 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, 1995, pp. 1-13.

[149] B. Chapman, P. Mehrotra, and H. Zima, "Extending HPF for Advanced Data-Parallel Applications," *IEEE Parallel and Distributed Technology*, Vol. 2, No. 3, 1994, pp. 15-27.

[150] T. F. Chen and J. L. Baer, "A Performance Study of Software and Hardware Data Prefetching Schemes," *Proc. of the 21st Ann. Int'l. Symp. on Computer Architecture*, April 1994, pp. 223-232.

[151] P. M. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson, "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys*, Vol. 26, June 1994, pp. 145-186.

[152] D. Y. Cheng, "A Survey of Parallel Programming Languages and Tools," *Technical Report RND-93-005*, NASA Ames Research Center, Moffett Field, Calif., 1993.

[153] C. Y. Chin and K. Hwang, "Packet Switching Networks for Multiprocesssors and Dataflow Computers," *IEEE Trans. Computers*, Nov. 1984, pp. 991-1003.

[154] F. T. Chong, B.-H. Lim, R. Bianchini, J. Kubiatowicz, and A. Agarwal, "Application Performance on the MIT Alewife Machine," *IEEE Computer*, Vol. 29, No. 12, Dec. 1996, pp. 57-64.

[155] Y. K. Chong and K. Hwang, "Evaluation of Four Consistency Models for Shared-Memory Multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, Oct. 1996, pp. 1085-1099.

[156] Chorus Systems, *CHORUS Kernel V3 R4.2 Specification and Interface*, CS/TR-91-69.1, 1993.

[157] C. K. Chow, "On Optimization for Storage Hierarchies," *IBM Journal of Research and Development*, 1974, pp. 194-203.

[158] A. Church and J.B. Rosser, "Some Properties of Conversion," *Trans. of the American Mathematical Society*, Vol. 39, 1936, pp. 472-482.

[159] R. Clark and K. Alnes, "An SCI Interconnect Chipset and Adaptor," *Symposium Records of Hot Interconnect IV*, 1996, pp. 221-235.

[160] J. Cocke and V. Markstein, "The Evolution of RISC Technology at IBM," *IBM Journal of Research and Development*, Vol. 34, No. 1, 1990, pp. 4-11.

[161] W. W. Collier, *Reasoning about Parallel Architecture*, Prentice-Hall, Englewood Cliffs, N.J., 1992.

[162] R. Colwell and R. Steck, "A Technical Tour of the Intel Pentium Pro Processor: A 0.6um BiCMOS Processor Employing Dynamic Execution," *ISSCC*, Feb. 1995.

[163] D. E. Comer, *Internetworking with TCP/IP*, 3d ed., Prentice Hall, Englewood Cliffs, N.J., 1995.

[164] Compaq , *Recovery Server Option Kit*, Compaq Computer Corp., http://www.compaq.com., 1997.

[165] Convex,, *CONVEX Exemplar Architecture*, CONVEX Press, Richardson, TX. 1994.

[166] Convex, *CONVEX Exemplar Programming Guide*, Order No. DSW-067, CONVEX Press, Richardson, TX. 1994.

[167] S.A. Cook, "A Taxonomy of Problems with Fast Parallel Algorithms," *Information and Control*, Vol. 64, pp. 2-22, 1985.

[168] Cornell Theory Center, "IBM RS/6000 Scalable POWERparallel System (SP)," 1995. http://www.tc.cornell.edu/UserDoc/Hardware/SP/.

[169] A. Cox, S. Dwarkadas, P. Keleher, B. Lu, R. Rajamony, and W. Zwaenepoel, "Software versus Hardware Shared-Memory Implementation: A Case Study," in *Proc. 21st Int'l. Symp. on Computer Architecture*, pp. 106-1117, April 1994.

[170] H. G Cragon, *Branch Strategy Taxonomy and Performance Models*, IEEE Computer Society Press, Los Alamitos, Calif., 1992.

[171] H. G. Cragon, *Memory Systems and Pipeline Processors*, Jones and Bartlett Publishers, Sudbury, Mass., 1996.

[172] D. Crawford, "ASCI Academia Strategic Alliances Program: Research Interests in Computer Systems and Computational and Computer Science Infrastructure," Sandia National Laboratories, December 1996.

[173] J.H. Crawford, "The i486 CPU: Executing Instructions in One Clock Cycle," *IEEE Micro*, 10(1):27-36, Feb. 1990.

[174] Cray Research, Inc. *Cray/MPP Announcement*, Eagan, Minn., 1992.

[175] Cray Research, Inc. Cray T3E Information, (http://www.cray.com/products/systems/crayt3e/), 1997.

[176] CSRD, "Perfect Club Benchmark Evaluation Package," Technical Report, Center for Supercomputer Research and Development, University of Illinois, Urbana, 1991.

[177] D. E. Culler et al. (NOW Project), "Efficient and Portable Implementation of MPI using Active Messages," University of California, Berkeley, (http://now.cs.berkeley.edu/Fastcomm/mpi.html), July 10, 1996.

[178] D. E. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation," *Proc. ACM Symp. on Principles and Practice of Parallel Programming*, 1993, pp. 1-12.

[179] D. E. Culler, K. Keeton, L.T. Liu, A. Mainwaring, R.P. Martin, S. Rodrigues, and K. Wright, *Generic Active Message Specification*, Version 1.1, Nov. 1994. Available at http://now.cs.berkeley.edu/.

[180] D. E. Culler, L.T. Liu, R.P. Martin, and C.O. Yoshikawa, "Assessing Fast Network Interfaces," *IEEE Micro*, Feb. 1996, pp. 35-43.

[181] D. E. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers, San Francisco, 1998.

[182] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D.A. Patterson, "Cooperative Caching: Using Remote Client Memory to Improve File System Performance," *Proc. of the 1st Symp. on Operating Systems Design and Implementation*, 1994, pp. 267-280.

[183] W. J. Dally, "Performance Analysis of *k*-ary *n*-Cube Interconnection Networks ," *IEEE Trans. Computers*, June 1990, pp. 775-785.

[184] W. J. Dally, "Network and Processor Architecture for Message-Driven Computers", in Suaya and Birtwistle(Eds.), *VLSI and Parallel Computation*, Chapter 3, Morgan Kaufmann, San Mateo, CA, 1990.

[185] W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans.on Computers*, C-36(5): 547-553, 1987.

[186] W. J. Dally, J. Fiske, J. Keen, R. Lethin, M. Noakes, P. Nuth, R. Davison, and G. Fyler, "The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanism," *IEEE Micro*, 12(2): 23-39, Apr. 1992.

[187] R. G. Daniels, "A Participant's Perspective," *IEEE Micro*, Vol. 16, No. 6, 1996, pp. 20-31.

[188] F. Darema, D. A. George, V.A. Norton, and G.F. Pfister, "A Single-Program-Multiple-Data Computational Model for EPEX Fortran," *Parallel Computing*, Vol. 7, 1988, pp. 11-24.

[189] H. Davis, S.R. Goldschmidt, and J. Hennessy, "Multiprocessor Simulation and Tracing Using Tango," *Proc. of Int'l. Conf. on Parallel Processing*, Vol. II, Aug. 1991, pp. II 99-107.

[190] J. R. Davy and P.M. Dew (Eds.), *Abstract Machine Models for Highly Parallel Computers*, Oxford Science Pub., Oxford, 1995.

[191] DEC, *Alpha Architecture Handbook*, Digital Equipment Corporation, Boxboro, MA., 1992.

[192] DEC, *Open VMS Clusters Handbook*, Document No. EC-H2207-93, Maynard, MA., 1993.

[193] DEC, *DEC OSF/1—Guide to DECthreads*, Part No. AA-Q2DPB-TK, Boxboro, MA., 1994.

[194] DEC, *AdvantageCluster: Digital's UNIX Cluster*, Sept. Boxboro, MA, 1994.

[195] DEC, *Digital Clusters for Windows NT*, http:/www.windowsnt.digital.com/clusters/default.htm, 1997.

[196] DEC, *Alpha 21164 Microprocessor Hardware Reference Manual*, Boxboro, MA., 1995.

[197] DEC, *TruCluster: Digital's UNIX Cluster*, Digital Part # EC-Z6310-43, Feb. 1996.

[198] DEC, *Digital's Unix Clusters Lead Industry in High Availability Commercial Solutions*, Maynard, MA., October, 1994.

[199] G. Deconinck, J. Vounckx, R. Cuyvers, and R. Lauwereins, "Survey of Checkpointing and Rollback Techniques," *Tech. Reports O3.1.8 and O3.1.12*, Katholieke Universiteit Leuven, June 1993.

[200] T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss, "Overview of the I-WAY: Wide Area Visual Supercomputing," *Int'l. Journal of Supercomputer Applications*, Vol. 10, No. 2, 1996.

[201] J. Del Rosario and A. Choudhary, "High-Performance I/O for Parallel Computers: Problems and Prospects," *IEEE Computer*, 27(3):59-68, 1994.

[202] M. Denman, P. Anderson, and M. Snyder, "Design of the PowerPC 604e Microprocessor," *Proc. of IEEE Compcon'96*, 1996, pp. 126-131.

[203] P. J. Denning, "Working Set Model for Program Behavior," *Comm. of ACM*, 11(6):323-333, 1968.

[204] S. Dickey, A. Gottlieb, and Y.-S. Liu, "Interconnection Network Switch Architectures and Combining Strategies," Ultracomputer Note #187, New York University, Sept. 1993.

[205] K. Diefendorff and P. K. Dubey, "How Multimedia Workloads Will Change Processor Design," *IEEE Computer*,Vol.30, No. 9, Sept. 1997, pp. 43-45.

[206] E.W. Dijkstra, "Solution of a Problem in Concurrent Programming Control", *Comm. ACM*, Vol. 8, Sept. 1965, pp.569-578.

[207] K. Dincer and G.C. Fox, "Building a World-Wide Virtual Machine Based on Web and HPCC Technologies," *Supercomputing'96 Conference Proceedings*, 1996.

[208] J. J. Dongarra, J. Martin, and J. Worlton, "Computer Benchmarking: Paths and Pitfalls," *IEEE Spectrum*, July 1986, p. 38.

[209] J. J. Dongarra, "The Linpack Benchmark-Parallel Report," http://performance.netlib.org/performance/html/linpack-parallel.data.co10.html, 1996.

[210] J. J. Dongarra and D. Walker, "Software Libraries for Linear Algebra Computations on High Performance Computers," *SIAM Review*, 1995.

[211] J. J. Dongarra, "The Performance Database Server (PDS): Reports: Linpack Benchmark - Parallel," http://performance.netlib.org/ performance/html/linpack-parallel.data.co10.html.

[212] J. J. Dongarra et al., "TOP 500 Supercomputers", http://www.netlib.org/benchmark/top500.html, June 20, 1996.

[213] X. Du and X. Zhang, "Coordinating Parallel Processes on Networks of Workstations," to appear in *Journal of Parallel and Distributed Computing*.

[214] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*, IEEE Computer Society Press, 1997.

[215] C. Dubnicki, A. Bilas, K. Li, and J.F. Philbin, "Design and Implementation of Virtual Memory-Mapped Communication on Myrinet," *Proc. of the 11th Int'l. Parallel Processing Symp.*, 1997.

[216] C. Dubnicki, L. Iftode, E.W. Felton, and K. Li, "Software Support for Virtual Memory-Mapped Communication," *Proc. of the 10th Int'l. Parallel Processing Symp.*, 1996.

[217] M. Dubois and C. Scheurich, "Memory Access Depndencies in Shared-Memory Multiprocessors," *IEEE Trans. Computers*, Vol. 16, No. 6, 1990. pp. 660-673.

[218] M. Dubois, C. Scheurich, and F.A. Briggs, "Memory Access Buffering in Multiprocessors," *Proc. 13th Int'l. Symp. on Computer Architecture*, 1986, pp. 434-442.

[219] M. Dubois, C. Scheurich and F. A. Briggs, "Synchronization, Coherence and Event Ordering in Multiprocessors," *IEEE Computer*, 21(2), 1988.

[220] M. Dubois and S. Thakkar (Eds.), *Scalable Shared-Memory Multiprocessors,* Kluwer Academic Publishers, Boston, Mass., 1992.

[221] M. Dubois and S. Thakkar (Eds.), *Cache and Interconnect Architecture in Multiprocessors,* Kluwer Academic Publishers, Boston, Mass., 1990.

[222] T. H. Dunigan; "Beta Testing the Intel Paragon MP," *Technical Report*, ORNL/TM-12830, Oak Ridge National Lab., June 1995.

[223] G. Eddon. *RRC for NT: Building Remote Procedure Calls for Windows NT Networks*, Prentice Hall, Englewood Cliffs, N.J., 1994.

[224] R. Edenfield, M. Gallup, W. Ledbetter, R. McGarity, E. Quintana, and R. Reininger, "The 68040 Processor: Part I. Design and Implementation," *IEEE Micro, Vol.* 10, No. 1, Feb. 1990, pp. 66-78.

[225] J. Edmondson, P. Rubinfeld, R. Preston, and V. Rajagopalan, "Superscalar Instruction Execution in the 21164 Alpha Microprocessor," *IEEE Micro*, April 1995, pp. 33-44.

[226] S. Eggers and R. Katz, "Evaluting the Performance of Four Snooping Cache Coherency Protocols," *Proc. of 16th Annual Int'l. Symp. on Computer Architecute*, May 1989, pp. 2-15.

[227] S. Eggers, J. S. Emer, H. Levy. J. Lo, R. Stamm, and D. Tullsen, "Simultaneous Multithreading: A Platform for Next-Generation Processors," *IEEE Micro*, September/October 1997.

[228] E. N. Elnozahy, D. B. Johnson, and Y. M. Wang, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," *Technical Report CMU-CS-96-181*, Department of Computer Science, Carnegie Mellon University, Sept. 1996.

[229] E. N. Elnozahy and W. Zwaenepoel, "Manetho: Transparent Rollback-Recovery with Low Overhead, Limited Rollback and Fast Output Commit," *IEEE Transactions on Computers*, Special Issue on Fault-Tolerant Computing, May 1992, pp. 526-531.

[230] H. El-Rewini, T. Lewis, and H.H. Ali, *Task Scheduling in Parallel and Distributed Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1994.

[231] D.E. Engerbresten, D.M. Kuchta, R.C. Booth, J.D. Crow, and W.G. Nation, "Parallel Fiber-Optic SCI Links," *IEEE Micro*, Vol. 16, No. 1, 1996, pp. 20-26.

[232] P. H. Enslow (ed.), *Multiprocessors adn Parallel Processing*, Wiley, New York, 1974.

[233] Executive Office of the President, Office of Science and Technololgy Policy, *A Research and Development Strategy for High Performance Computing*, November 1987.

[234] F. Faggin, M.E. Hoff, Jr., S. Mazor, and M. Shima, "The History of the 4004," *IEEE Micro*, Vol. 16, No. 6, 1996, pp. 10-20.

[235] R. Fatoohi, "Performance Evaluation of Communication Networks for Distributed Computing," NASA Ames Research Center Report NAS-95-009, 1995.

[236] D. G. Feitelson and L. Rudolph (Eds.), *Job Scheduling Strategies for Parallel Processing*, LNCS 949, Springer-Verlag, Berlin, 1995.

[237] E. W. Felten, R.D. Alpert, A. Bilas, M.A. Blumrich, D.W. Clark, S.N. Damianakis, C. Dubnicki, L. Iftode, and K. Li, "Early Experience with Message-Passing on the SHRIMP Multicomputer," *Proc. the 23rd Int'l. Symp. on Computer Architecture*, 1996.

[238] T. Y. Feng, "A Survey of Interconnection Networks," *IEEE Computer*, Vol. 14. No. 12, 1981, pp. 12-27.

[239] D. M. Fenwick, D.J. Foley, W.B. Gist, S. R. VanDoren, and D. Wissel, "The AlphaServer 8000 Series: High-End Server Platform Development," *Digital Technical Journal*, Vol. 7, No. 1, 1995, pp. 43-65.

[240] J. A. Fisher, "Trace Scheduling: A Technique for Global Microcode Compaction," *IEEE Trans. Computers*, Vol. 30, No. 7, 1981, pp. 478-490.

[241] J. A. Fisher, "Very Long Instruction Word Architectures and the ELI-512," *Proc. 10th Annual Symp. Computer Arch.*, ACM Press, New York, 1983, pp. 140-150.

[242] J. A. Fisher, "Walk-Time Techniques: Catalyst fro Architectural Change," *IEEE Computer Magazine*, Vol.30, No. 9, Sept. 1997, pp. 40-42.

[243] J. L. Flanagan, "Technologies for Mutlimedia Communications," *Proceedings of the IEEE*, Vol. 82, No. 4, April 1994, pp. 590-603.

[244] M. J. Flynn, "Some Computer Organizations and Their Effectiveness," *IEEE Trans. on Computers*, Vol. C-21, 1972, pp. 948-960.

[245] M. J. Flynn, *Computer Architecture: Pipelined and Parallel Processor Design*, Jones and Bartlett, Boston, Mass., 1995.

[246] S. Fortune and J. Wyllie, "Parallelism in Random Access Machines," *Proc. ACM Symp. on Theory of Computing*, 1978, pp. 114-118.

[247] I. Foster and K. M. Chandy, "Fortran M: A Language for Modular Parallel Programming," *J. Parallel and Distributed Computing*, Vol. 25, No. 1, Jan. 1995.

[248] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke, "Software Infrastructure for the I-WAY High-Performance Distributed Computing Experiment," *Proc. of the 5th IEEE Symp. on High Performance Distributed Computing*, 1996.

[249] I. Foster, J. Geisler, C. Kesselman, and S. Tuecke, "Multimethod Communication for High-Performance Metacomputing Applications," *Supercomputing '96 Conference Proceedings*, 1996.

[250] I. Foster, C. Kessekman, and S. Tuecke, "The Nexus Approach to Integrating Multithreading and Communication," *Journal of Parallel and Distributed Computing*, Vol. 37, No. 1, 1996, pp. 70-82.

[251] G. Fox and W. Furmanski, "Towards Web/Java Based High Performance Distributed Computing - An Evolving Virtual Machine," *Proc. of the 5th IEEE Symp. on High Performance Distributed Computing*, August 1996.

[252] G. Fox, W. Furmanski, M. Chen, C. Rebbi and J. Cowie, "WebWork: Integrated Programming Environment Tools for National and Grand Challenges," *NPAC Tech. Report SCCS-0715*, Syracuse University, 1995.

[253] G. C. Fox, M.A. Johnson, G.A. Lyzenga, S.W. Otto, J.K. Salmon, and D.W. Walker, *Solving Problems on Concurrent Processors*, Vol. I, Prentice-Hall, Englewood Cliffs, N.J., 1988.

[254] G. C. Fox, R.D. Williams, and P.C. Messina, *Parallel Computing Works!*, Morgan Kaufmann Publishers, San Francisco, 1994.

[255] A. G. Fraser, "Future WAN Telecommunications," *IEEE Micro*, Vol. 16, No. 1, 1996, pp. 53-57.

[256] D. Frye, Ray Bryant, H. Ho, R. Lawrence, and M. Snir. "An External User Interface for Scalable Parallel Systems," *Technical Report*, International Businesss Machines, Armonk, N.Y., May 1992.

[257] E. M. Frymoyer, "Fibre Channel Fusion: Low Latency, High Speed," *Data Communications*, http://www.data.com/, Feb. 1995.

[258] D. D. Gajski and J. K. Peir, "Essential Issues in Multiprocessor Systems," *IEEE Computer*, Vol. 18, June, 1985.

[259] J. Gary, and D.P. Siewiorek, "High-Availability Computer Systems," *IEEE Computer*, Sept. 1991, pp. 39-48.

[260] G. A. Geist and V. S. Sunderam, "The Evolution of the PVM Concurrent Computing System," *Proceeding of the 26th IEEE Compcon Symposium*, pages 471-478, San Francisco, February 1993.

[261] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, and V. Sunderam, *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, Mass., 1994.

[262] A.V. Gerbessiotis and L.G. Valiant, "Direct Bulk-Synchronous Parallel Algorithms," *J. Parallel and Distributed Computing*, Vol. 22, No. 2, Feb. 1994, pp. 251-267.

[263] D. Gelernter, "Generative Communication in Linda," *ACM Trans. Program Lang. and Syst.*, 1985, 7(1): 80-112.

[264] K. Gharachorloo et al., "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors," *Proc. of the 17th Annual Symp. on Computer Architecture*, June 1990, pp. 15-25.

[265] K. Gharachorloo, S. Adve, A. Gupta, J. L. Hennessy, and M. Hill, "Programming for Different Memory Consistency Models," *Journal of Parallel and Distributed Computing*, Aug. 1992.

[266] G. A. Gibson, *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*, MIT Press, Cambridge, Mass., 1992.

[267] Gigabit Ethernet Alliance, "White Paper of Gigabit Ethernet", March 1997. Also avaialble from web site http ://www.gigabit-ethernet.org/.

[268] R. B. Gillett and R. Kaufmann, "Using the Memory Channel Networks," *IEEE Micro*, Jan./Feb., 1997, pp. 19-25.

[269] L. M. Goldschlager, "A Universal Interconnection Pattern for Parallel Computers," *Journal of ACM*, Vol. 29, 1982, pp. 1073-1086.

[270] S. C. Goldstein, K.E. Schauser, and D.E. Culler, "Lazy Threads: Implementing a Fast Parallel Call," *Journal of Parallel and Distributed Computing*, Vol. 37, No. 1, 1996, pp. 5-20.

[271] H. Goldstine and J. von Neumann, "On the Principles of Large-Scale Computing Machines," *Collected Works of John von Neumann*, Vol. 5. Pergamon, New York, 1963.

[272] J. R. Goodman, M. Vernon, and P. Woest, "Efficient Synchronization Primitives for Large-Scale Cache-Coherent Multiprocessors," *Proc. Third Int'l. Conf. on Architectural Support for Programming Languages and Operating systems*, April 1989, pp. 64-73.

[273] J. R. Goodman, "Cache Consistency and Sequential Consistency," *Technical Report 61*, IEEE SCI Committee, 1990.

[274] B. Groop, R. Lusk, T. Skjellum, and N. Doss; "Portable MPI Model Implementation," Argonne National Laboratory, *Technical Report*, July 1994.

[275] J. C. Gomez, V. Rego, and V.S. Sunderam, "Efficient Multithreaded User-Space Transport for Network Computing: Design and Test of the TRAP Protocol," *Journal of Parallel and Distributed Computing*, Vol. 40, No. 1, 1997, pp. 103-117.

[276] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer: Designing a MIMD, Shared Memory Parallel Computer," *IEEE Trans.on Computers*, C-32(2):175-189, 1983.

[277] H. Grahn and P. Stenstrom, "Efficient Strategies for Software-Only Directory Protocols in Shared Memory Multiprocessors," *Proc. of the 22nd Annual Int'l. Symp. on Computer Architecture*, 1995.

[278] A.Y. Grama, A. Gupta, and V. Kumar, "Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures," *IEEE Parallel & Distributed Technology*, 1(3):12-21, 1993.

[279] J. Gray (Ed.), *The Benchmark Handbook for Database and Transaction Processing Systems*, Morgan Kaufmann, San Mateo, Calif., 1991.

[280] B. Grayson, A.P. Shah, and R.A. van de Geijn, "A High Performance Parallel Stranssen Implementation," *Tech. Report*, Dept. of Computer Science, University of Texas at Austin, 1995.

[281] D. Greenley et al., "UltraSPARC: The Next Generation Superscalar 64-bit SPARC," *Digest of Papers, Compcon*, Spring 1995, pp. 442-451.

[282] A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. Reynolds, Jr., "Legion: The Next Logical Step towards a Nationwide Virtual Computer," *Tech. Report CS-94-21*, Department of Computer Science, University of Virginia, 1994.

[283] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, MIT Press, Cambridge, Mass., 1994.

[284] N. J. Gunther, "Issues Facing Commercial OLTP Applications on MPP Platforms," *Digest of Papers, IEEE Compcon'94*, pp. 242-247.

[285] A. Gupta, J. Hennessy, K. Gharachorloo, T. Mowry, and W.-D. Weber. "Comparative Evaluaton of Latency Reducing and Tolerating Techniques," *Proc. 18th Int'l. Symp. on Computer Architecture*, June 1991, pp. 254-263.

[286] J. L. Gustafson, "Reevaluating Amdahl's Law," *Comm. of ACM*, Vol. 31, No. 5, 1988, pp. 532-533.

[287] J. L. Gustafson and Q.O. Snell, "HINT: A New Way to Measure Computer Performance," *Proc. of Supercomputing'96*, 1996.

[288] D. B. Gustavson, "The Scalable Coherent Interface and Related Standards Projects," *IEEE Micro*, Vol. 12, No. 2, 1992, pp. 10-12.

[289] D. B. Gustavson and Q. Li, "Low Latency, High-Bandwidth, and Low Cost for Local-Area Multi-Processor," Dept. of Computer Engineering, Santa Clara University, 1996.

[290] L. Gwennap, "Intel's P6 Uses Decoupled Superscalar Design," *Microprocessor Report,* February 1995, pp. 5-15.

[291] L. Gwennap, "MIPS R10000 Uses Decoupled Architecture," *Microprocessor Report*, October 1994, pp. 18-22.

[292] M. W. Hall, J.M. Anderson, S.P. Amarasinghe, B.R. Murphy, S.W. Liao, E. Bugnnion, and M.S. Lam, "Maximizing Multiprocessor Performance with the SUIF Compiler," *IEEE Computer*, Vol. 29, No. 12, Dec. 1996, pp. 84-89.

[293] E. Hagersten, A. Landin, and S. Haridi, "Multiprocessor Consistency and Synchronization Through Transient Cache States," in Dubois and Thakkar (Eds.), *Scalable Shared-Memory Multiprocessors,* Kluwer Academic Publisheds, Boston, MA, 1992.

[294] E. Hagersten, A. Landin, and S. Haridi, "DDM - A Cache-Only Memory Architecture," *IEEE Computer,* 25(9):44-54, Septemeber 1992.

[295] F. Halsall, *Data Communications, Computer Networks and Open Systems* (Fourth Edition), Addison-Wesley, Reading, Mass., 1996.

[296] L. Hammond, B. Nayfeh, and K. Olukotun, "A Single-Chip Multiprocessor," *IEEE Computer Magazine*,Vol.30, No. 9, Sept. 1997, pp. 79-85.

[297] T. J. Harris and N.P. Topham, "The Scalability of Decoupled Multiprocessors," *Proc. of Scalable High-Performance Computing Conference*, Knoxville, Tenn., May 1994, pp. 17-22.

[298] J. Harris, J.A. Bircsak, M.R. Bolduc, J.A. Diewald, I. Gale, N.W. Johnson, S. Lee, C.A. Nelson, and C.D. Offner, "Compiling High Performance Fortran for Distributed-Memory Systems," *Digital Technical Journal*, Vol. 7, No. 3, 1995, pp. 5-23.

[299] P. Hatcher and M. Quinn, *Data-Parallel Programming on MIMD Computers*, MIT Press, 1991.

[300] A. H. Hayes, M.L. Simmons, J.S. Brown, and D.A. Reed (Eds.), *Debugging and Performance Tuning for Parallel Computing Systems*, IEEE Computer Society Press, July 1996.

[301] M. Heath and J. Etheridge, "Visualizing the Performance of Parallel Programs," *IEEE Software*, 8(5):29-39, 1991.

[302] J. Heinrich, *MIPS R10000 Microprocessor User's Manual,* MIPS Technologies, Inc., 1994.

[303] R. Hempel, "The ANL/GMD Macros (PARMACS) in Fortran for Portable Parallel Programming Using the Message Passing Model: Users Guide and Reference Manual," Gesellschaft fur Mathematik und Datenverarbeitung (GMD) mbH, 1991.

[304] M. Henderson, B. Nickless, and R. Stevens, "A Scalable High-Performance I/O System," *Proc. 1994 Scalable High-Performance Computing Conf.*, pp. 79-86. IEEE Computer Society, 1994.

[305] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach,* Morgan Kaufmann, San Mateo, CA, 1995.

[306] M.P. Herlihy, "Wait-Free Synchronization," *ACM Trans. on Program Lang. and Syst*, 1990, 12(3):463-492.

[307] M.P. Herlihy and J.E.B. Moss, "Transactional Memory: Architectural Support for Lock-Free Data Structures," *Proc. of the 20th Int'l. Symp. Computer Architecture*, pp. 289-300, 1993.

[308] M. Hill, "What Is Scalability?," ACM *Computer Architecture News*, Dec. 1990, pp 18-21.

[309] W. D. Hillis, "The Connection Machine," *Scientific American*, Vol. 256, June 1987, pp. 108-115.

[310] W.D. Hillis and G. L. Steele, "Data Parallel Algorithms," *Comm. ACM*, 29(12):1170-1183, 1986.

[311] S. Hiraanandani, K. Kennedy, and C.W. Tseng, "Compiling Fotran D for MIMD Disytributed Machines," *Comm.of the ACM*, Aug.1992, pp. 66-80.

[312] C. A. Hoare, *Communicating Sequential Processes,* Prentice-Hall, London, U.K. , 1984.

[313] R. W. Hockney, "Performance Parameters and Benchmarking of Supercomputers," *Parallel Computing*, Vol. 17, pp. 1111-1130, 1991.

[314] R. W. Hockney, "The Communication Challenge for MPP: Intel Paragon and Meiko CS-2," *Parallel Computing, Vol.* 20, 1994, pp. 389-398.

[315] R. W. Hockney, "A Framework for Benchmark Performance Analysis," in J.J. Dongarra and W. Gentzsch (Eds.), *Computer Benchmarks, Advances in Parallel Computing*, Vol. 8, Elsevier Science Pub., 1993, pp. 65-76.

[316] R. W. Hockney, *The Science of Computer Benchmarking*, The Society for Industrial and Applied Mathematics, Philadelphia, 1996.

[317] R. W. Hockney and M. Berry, "Public International Benchmarks for Parallel Computers: PARKBENCH Committee Report No. 1," *Scientific Computing*, 3(2):101-146, 1994.

[318] R. W. Hockney and C. R. Jesshope, *Parallel Computers: Architecture, Programming, and Algorithms*, Adam Hilger, Philadelphia, 1988.

[319] C. Holt, J. P. Singh, and J. Hennessy, "Application and Architectural Bottlenecks in Large Scale Distributed Shared Memory Machines," *Proc. of the 23rd Int'l. Symp. on Computer Architecture*, 1996, pp. 134-145.

[320] R. W. Horst, "Massively Parallel Systems You Can Trust," *Digest of Papers, IEEE Compcon'94*, pp. 236-241.

[321] Hotchips, *Proc. Hot Chips III Symp. on High-Performance Chips*, Stanford University, Palo Alto, CA, 1991.

[322] D. Hunt, "Advanced Features of the 64-bit PA-8000," *Proceedings 1995 IEEE COMPCON*, 1995, pp. 123-128.

[323] K. Hwang, W. Croft, G Goble, B. W. Wah, F. A. Briggs, W. Simmons, and C. L. Coates, "A UNIX-Based Local Computer Network with Load Balancing," *IEEE Computer*, April 1982, pp. 55-66.

[324] K. Hwang, "Advanced Parallel Processing with Supercomputer Architectures," *Proc. of the IEEE*, 1987, pp. 1348-1379.

[325] K. Hwang, P. S. Tseng, and D. Kim, "An Orthogonal Multiprocessor for Parallel Scientific Computations," *IEEE Trans. Computers*, Vol. C-38, No. 1, Jan. 1989, pp. 47-61.

[326] K. Hwang, "Exploiting Parallelism in Multiprocessors and Multicomputers," in *Parallel Processing for Supercomputers and Artificial Intelligence*, K. Hwang and D. DeGroot (Eds.), McGraw-Hill, New York, NY, 1989, pp. 1-68.

[327] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, and Programmability*, McGraw-Hill, New York, 1993.

[328] K. Hwang, "Gigabit Networks for Building Scalable Multiprocessors and Multicomputer Clusters," *HKIE Transactions* , Hong Kong Institute of Engineers, Hong Kong, 1997.

[329] K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, New York, 1983.

[330] K. Hwang and D. DeGroot (Eds.), *Parallel Processing for Supercomputers and Artificial Intelligence*, McGraw-Hill, New York, 1989.

[331] K. Hwang and J. Ghosh, "Hypernet: A Communication-Efficient Architecture for Multicomputers," *IEEE Trans. on Computers*, Nov. 1989, pp. 1450-1466.

[332] K. Hwang, C. J. Wang, and C.-L. Wang, "Evaluating MPI Collective Communication on the SP2, T3D, and Paragon Multicomputers," *IEEE High-Performance Computer Architecture* (HPCA-3), San Antonio, Texas, Feb. 1-5, 1997.

[333] K. Hwang and Z. Xu, "Scalable Parallel Computers for Real-Time Signal Processing," *IEEE Signal Processing*, Vol. 13, No. 4, 1996, pp. 50-66.

[334] K. Hwang, Z. Xu, and M. Arakawa, "Benchmark Evaluation of the IBM SP2 for Parallel Signal Processing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 5, 1996, pp. 522-536.

[335] IBM Corp., *RISC System/6000 Technology*, IBM Advanced Workstations Division, Austin, TX, 1990.

[336] IBM Corp., *LoadLeveller General Information Manual*. Document GH26-7227, Armonk, NY.

[337] IBM Corp., *AIX Parallel Environment: Programming Primer*, Release 2.0, Pub. No. SH26-7223, June 1994.

[338] IBM Corp. "*High Availability Cluster Multiprocessing/6000 (HACMP) System Overview,*" Doc. No. SC23-2408-02, Armonk, N.Y., 1993.

[339] IBM Corp., *Sysplex Hardware and Software Migration*, Doc. No. GC 28-1210-00, Armonk, N.Y., 1994.

[340] IBM Corp., *AIX 4.1 Threads*, Armonk, N.Y., 1995.

[341] IBM Corp., *IBM System/370 Principles of Operation*, IBM Pub. GA22-7000-9, File S370-01, 1983.

[342] L. Iftode, C. Dubnicki, E.W. Felten, and K. Li, "Improving Release-Consistent Shared Virtual Memory Using Automatic Update," *Proc. 2nd Int'l. Symp. on High-Performance Computer Architecture*, 1996.

[343] IEEE, *ANSI/IEEE Standard 1596-1992: Scalable Coherent Interface*, IEEE, Piscataway, N.J., 1993; and *Standard 1596.3-1996*, 1996.

[344] IEEE, *Futurebus+: Logical Layer Specifications 896.1-1991*, Microprocessor Standards Subcommittee, IEEE Computer Society, 1991.

[345] IEEE, *POSIX P1003.4a: Threads Extension for Portable Operating Systems*, IEEE, Piscataway, N.J., 1994.

[346] Intel Corp., Material for Standard High Volume (SHV) servers can be found on Intel's Web sites, such as http://www.intel.com/intel/june297/foils/grove/index.htm, http://www.intel.com/intel/june297/foils/miner/, and http://www.intel.com/procs/SERVERS/feature/shv/

[347] Intel/Sandia , *The ASCI Teraflop Machine*, http://abacus.ssd.intel.com/tflop1.html, 1996.

[348] D. V. James et al., "Distributed-Directory Scheme: Scalable Coherence Interface," IEEE Computer, June 1990, pp. 74-77.

[349] E. H. Jensen, G.W. Hagensen, and J.M. Broughton, "A New Approach to Exclusive Data Access in Shared-Memory Multiprocessors," *Technical Report UCRL-97663*, Lawrence Livermore National Laboratory, 1987.

[350] T. Jermoluk, *Multiprocessor UNIX*, Silicon Grpahics Inc., Santa Clara, CA, 1990.

[351] D. Johnson, D. Lilja, J. Riedl, and J. Anderson, "Low-Cost, High-Performance Barrier Synchronization on Networks of Workstations," *Journal of Parallel and Distributed Computing*, Vol. 40, No. 1, 1997, pp. 118-130.

[352] M. Johnson, *Superscalar Microprocessor Design*, Prentice-Hall, Englewood Cliffs, N.J., 1991.

[353] L. Johnsson, "Communication in Network Architectures," in *VLSI and Parallel Computation*, Morgan Kaufmann Publishers, San Mateo, CA. 1990.

[354] L. Johnsson and C.-T. Ho, "Matrix Transposition on Boolean *n*-Cube Configured Ensemble Architectures," *SIAM J. Matrix Analysis and Applications*, 9(3):419-454, 1988.

[355] L. Johnsson and C.-T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes," *IEEE Trans. Computs.*, C-38(9):1249-1268, 1989.

[356] N. P. Jouppi and D. W. Wall, "Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines," *Proc. Third Int'l. Conf. Arch. Support for Prog. Lang. and OS*, pp. 272-282, ACM Press, New York, 1989.

[357] D. Kafura and L. Huang, "MPI++: A C++ Language Binding for MPI," *MPI Developers Conference*, University of Notre Dame, June 1995.

[358] A. Kagi, D. Burger, and J.R. Goodman, "Efficient Synchronization: Let Them Eat QOLB," *Proc. of the 24th Annual Int'l. Symp. on Computer Architecture*, 1997, pp. 170-180.

[359] G. Kane, *MIPS R2000 RISC Architecture*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[360] V. Karamcheti and A. A. Chien, "Software Overhead in Messaging Layers: Where Does the Time Go?" *Proc. 6th Inl. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 51-60, 1994.

[361] V. Karamcheti and A.A. Chien, "A Comparison of Architectural Support for Messaging in the TMC CM-5 and the Cray T3D," *Proc. 22nd Int'l. Symp. on Computer Architecture*, pp. 298-307, 1995.

[362] A. H. Karp and R. G. Babb II, "A Comparison of 12 Parallel Fortran Dialects," *IEEE Software*, Vol.5, No.5, 1988, pp. 52-66.

[363] A. H. Karp, "Programming for Parallelism," *IEEE Computer*, Vol. 20, No. 5, 1987, pp. 43-57.

[364] A. H. Karp and H. P. Flatt, "Measuring Parallel Processor Performance," *Comm. of the ACM*, May, 1990, pp. 539-543.

[365] W. J. Kaufmann and L. L. Smarr, *Supercomputing and the Transformation of Science*, Scientific American Library, 1993.

[366] R. M. Keller, "Formal Verification of Parallel Programs," *Comm. of ACM*, Vol.19, July 1976, pp. 371-384.

[367] K. Kennedy, "Compiler Technology for Machine-Independent Parallel Programming," *Int'l. Journal of Parallel Programming*, Vol. 22, No. 1, 1994, pp. 79-98.

[368] B. Kernighan and D. Ritchie, *The C Programming Language*, 2d ed., Prentice Hall, 1988.

[369] Y. A. Khalidi, J. M. Bernabeu, V. Matena, K. Shirriff, and M. Thadani, "Solaris MC: A Multicomputer OS," Sun Microsystems Lab. SMLI TR-95-48, Nov. 1995. Also appeared in *Proc. of the 1996 USENIX Conference*.

[370] C. Koelbel, D. Loveman, R. Schreiber, G. Steele, and M. Zosel. *The High Performance Fortran Handbook*, MIT Press, 1994.

[371] R. K. Koeninger, M. Furtney, and M. Walker, "A Shared Memory MPP from Cray Reseach," *Digital Technical Journal*, Vol. 6, No. 2, Spring 1994, pp. 8-21.

[372] A. Kolawa, "Parasoft: A Comprehensive Approach to Parallel and Distributed Computing," *Proc. Workshop on Cluster Computing*, 1992.

[373] L. Kontothanassis et al., "VM-Based Shared Memory on Low-Latency, Remote-Memory-Access Networks", *Proc. of the 24th Int'l. Symp. Computer Architecture*, June 1997, pp. 157-169.

[374] L. I. Kontothanassis, R.W. Wisniewski, and M.L. Scott, "Scheduler-Conscious Synchronization," *ACM Trans. on Computer Systems*, Vol. 15, No. 1, 1997, pp. 3-40.

[375] R. B. Konuru, S.W. Otto, and J. Walpole, "A Migratable User-Level Process Package for PVM," *Journal of Parallel and Distributed Computing*, Vol. 40, No. 1, 1997, pp. 81-102.

[376] J. S. Kowalik (Ed.), *Parallel MIMD Computation: HEP Supercomputer and Applications*, MIT Press, Cambridge, MA, 1985.

[377] C. E. Kozyrakis, S. Perissakis, D. Patterson, T. Anderson, K. Asanovic, N. Cardwell, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, R. Thomas, N. Treuhaft, and K. Yelick, "Scalable Processors in the Billion-Transistors Era: IRAM," *IEEE Computer*, Vol.30, No. 9, Sept. 1997, pp. 75-78.

[378] N. P. Kronenberg, H. M. Levy, W. D. Strecker, and R. J. Merewood. "The VaxCluster Concept: An overview of a Distributed System". *Digital Technical Journal*, Vol. 4, No. 7, Sept. 1987, pp. 15-21.

[379] C. P. Kruskal and A. Weiss, "Allocating Independent Subtasks on Parallel Processors," *IEEE Trans. Software Engineering*," Vol. SE-11, pp. 1001-1016, 1985.

[380] C.P. Kruskal, L. Rudolph, and M. Snir, "Efficient Synchronization of Multiprocessors with Shared Memory," *ACM Trans. Program. Lang. Systems*, April, 1988, pp. 579-601.

[381] Kuck and Associates, *The KAP Preprocessor*, http://www.kai.com/kap/kap_what_is.html.

[382] D.J. Kuck, E.S. Davidson, D.H. Lawrie, and A.H. Sameh, "Parallel Supercomputing Today - The Cedar Approach," *Science*, 231(2), Feb. 1986.

[383] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing*, Benjamin/ Cummings, New York, 1993.

[384] V. Kumar, A. Grama, and V. Rao, "Scalable Load Balancing Techniques for Parallel Computers," *J. Parallel and Distributed Computing*, 22(1):60-79, 1994.

[385] H. T. Kung and C. E. Leiserson, "Systolic Arrays for VLSI " in *SIAM Sparse Matrix Proceedings*, edited by Duff and Stewart, Knoxville, Tenn., 1978.

[386] H. T. Kung el.al., "Network-Based Multicomputers: An Emerging Parallel Architecture," *Proc. of Supercomputing*, IEEE Computer Society Press, 1991, pp. 664-673.

[387] H. T. Kung, "Gigabit Local Area Networks: A Systems Perspective," *IEEE Communicatins Magazine*, April, pp. 79-89.

[388] J. Kuskin et al., "The Stanford FLASH Multiprocessor," *Proc. 21st Int. Symp. on Computer Architecture*, April 1994, pp. 302-313.

[389] M. S. Lam, "Software Pipelining: An Effective Scheduling Technique for VLIW Machines," *Proc. ACM SIGPLAN Conf. Prog. Lang, Design and Implementation*, 1988, pp. 318-328.

[390] L. Lamport, "Solved Problems, Unsolved Problems and Non-Problems in Concurrency," *Proc. of Int'l. Conference on Distributed Computing*, 1983, pp. 1-11.

[391] L. Lamport, "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs," *IEEE Trans. Computers*, Vol. C-28, Sept. 1979, pp. 690-691.

[392] J. R. Larus, "Loop-Level Parallelism in Numeric and Symbolic Programs," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 4, No. 7, 1993, pp. 812-826.

[393] J. Laudon and D. Lenoski, "The SGI Origin: A ccNUMA Highly Scalable Server," *Proc. of the 24th Int'l. Symp. Computer Architecture*, June 1997, pp. 241-251.

[394] M. Lauria and A. Chien, "MPI-FM: High Performance MPI on Workstation Clusters," *Journal of Parallel and Distributed Computing*, Vol. 40, No. 1, 1997, pp. 4-18.

[395] D. H. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Trans. on Computers*, Dec. 1975.

[396] H. Lawson, *Parallel Processing in Industrial Real-time Applications*, Prentice Hall, N.J., 1992.

[397] J. V. Lawton, J.J. Brosnan, M.P. Doyle, and S.D. Riodain, "Building a High-Performance Message-Passing System for MEMORY CHANNELClusters," *Digital Technical Journal*, Vol. 8, No. 2, 1996, pp. 96-116.

[398] R. B. Lee, "Precision Architecture," IEEE *Computer*, Vol. 22, No. 1, Jan. 1989, pp. 78-91.

[399] R. Lee, P. C. Yew, and D. Lawrie, "Data Prefetching in Shared Memory Multiprocessors", *Proc. Int'l. Conf. on Parallel Processing*, August 1987, pp. 28-31.

[400] R. B. Lee, "Subword Parallelism with MAX-2," *IEEE Micro*, Vol. 16, No. 4, 1996, pp. 51-59.

[401] R. B. Lee and M. D. Smith, "Media Processing: A New Design Target," *IEEE Micro*, August, 1996, pp. 6-9.

[402] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann, 1992.

[403] C. Leiserson, "Fat Tree: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Trans. on Computers*, Oct. 1985, pp. 892-901.

[404] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hills, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S. Yang, and R. Zak, "The Network Architecture of the CM-5," *Int'l. Symp. on Parallel and Distributed Algorithms '92*, June 1992, pp. 272-285.

[405] D. E. Lenoski et al., "The DASH Prototype: Logic and Performance," *IEEE Trans. on Parallel and Distributed Systems*, Jan. 1993, pp. 41-61.

[406] D. E. Lenoski and W.-D. Weber, *Scalable Shared-Memory Multiprocessing*, Morgan Kaufmann, San Francisco, CA, 1995.

[407] E. Levin, "Grand Challenges in Computational Science," *Comm. of the ACM*, Vol. 32, No. 12, Dec. 1989, pp. 1456-1457.

[408] D. Levitan, T. Thomas and P. Tu, "The PowerPC 620 Microprocessor: A High Performance Superscalar RISC Microprocessor," *Digest of Papers, Compcon95*, Spring 1995, pp. 285-291.

[409] E. Levy and A. Silberschartz, "Distibuted File Systems: Concepts and Examples," *ACM Computing Surveys*, Vol. 22, No. 4, 1990, pp. 321-374.

[410] M. J. Lewis and A. Grimshaw, "The Core Legion Object Model," *Proc of the Fifth IEEE Int'l. Symp. on High Performance Distributed Computing*, IEEE Computer Society Press, August 1996.

[411] T. G. Lewis and H. El-Rewini, *Introduction to Parallel Computing*, Prentice-Hall, N.J., 1992.

[412] T. G. Lewis, "The Nethead Gang," *IEEE Computer*, 1995, 28(12): 8-10.

[413] K. Li, "IVY: A Shared-Virtual Memory System for Parallel Computing," *Proc. Int'l. Conf. Parallel Processing*, August 1988, pp. 94-101

[414] K. Li and P. Hudak, "Memory Coherence in Shared Virtual Memory Systems," *ACM Trans. Computer Systems*, Nov. 1989, pp. 321-359.

[415] K. Li, J.F. Naughton, and J.S. Plank, "Low-Latency, Concurrent Checkpointing for Parallel Programs," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, No. 8, 1994, pp. 874-879.

[416] D. J. Lilja, *Architectural Alternatives for Exploiting Parallelism*, IEEE Computer Society Press, Los Alamitos, CA, 1992.

[417] X. Lin and L. M. Ni, "Deadlock-Free Multicast Wormhole Routing in Multicomputer Networks," *Proc. 18th Annu. Int'l. Symp. Computer Arch.*, pp. 116-125, 1991.

[418] M. H. Lipasti and J. P. Shen, "Superspeculative Microarchitecture for Beyond AD 2000," *IEEE Computer*, Vol.30, No. 9, Sept. 1997, pp. 59-66.

[419] D. Loveman, "High Performance Fortran," *IEEE Parallel and Distributed Technology*, 1(1):25-42, 1993.

[420] T. Lovett and R. Clapp, "STiNG: A CC-NUMA Computer System for the Commercial Marketplace," *Proc. of the 23rd Annual Int'l. Symp. on Computer Architecture*, 1996, pp. 308-317.

[421] T. D. Lovett, R. M. Clapp and R. J. Safranek, "*NUMA-Q: An SCI-Based Enterprise Server*," Sequent Computer White Paper, http://www.sequent.com/, 1996.

[422] E. Lusk et al., *Portable Programs for Parallel Processors*, Holt, Rinehardt, and Winston, New York, 1987.

[423] R. Lusk, N. Doss, A. Skjellum, and W. Gropp; "MPICH: A High-Performance Portable Implementation of the MPI Standard," *MPI Developers Conference*, University of Notre Dame, June 1995.

[424] S. Mahlke, R. Hank, J. McCormick, D. August, and W.M. Hwu, "A Comparison of Full and Partial Predicated Execution Support for ILP Processors," *Proc. of the 22nd Annual Int'l. Symp. on Computer Architecture*, June 1995, Italy, pp. 138-149.

[425] Marathon Corp., *MIAL1/2 Fault-Tolerant Cluster*, Web: http://www.mial.com, 1997.

[426] A. Mainwaring and D.E. Culler, "Generic Active Message Applications Programming Inteface and Communication Subsystems Organization," Division of Computer Science, University of California at Berkeley, Nov. 1996. Available at http://now.cs.berkeley.edu/.

[427] J. MarcFrailong et al., "The Next Generation SPARC Multiprcoessing System Architecutre," *Proceedings of COMPCON*, Spring 1993, pp. 475-480.

[428] N. Margulis, *i860 Microprocessor Architecture*, Intel Osborne/McGraw-Hill, Berkeley, CA, 1990.

[429] J. Markoff, "The Microprocessor's Impact on Society," *IEEE Micro*, Vol. 16, No. 6, 1996, pp. 54-59.

[430] M. A. Marson, G. Balbo, and G. Conte, *Performance Modules of Multiprocessor Systems*, MIT Press, Cambridge, MA, 1988.

[431] R. P. Martin et al., "Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture," *Proc. of the 24th Int'l. Symp. Computer Architecture*, June 1997, pp. 85-96.

[432] J. R. Mashey, "From Buses to Modular, Distributed Crossbars: Not All Shared-Memory Systems Are Equal," Silicon Graphics, 1997. Available at http://www.sgi.com/Prooducts/hardware/servers/techtalk.html.

[433] T. G. Mattson, D. Scott, and S. Wheat, "A TeraFLOPS Supercomputer in 1996: The ASCI TFLOPS System," *Proc. of the 6th Int'l. Parallel Processing Symp.*, 1996, pp. 84-93.

[434] C. May et al. (Eds.), *The PowerPC Architecture: A Specification for a New Family of RISC Processors*, Morgan Kaufmann, San Francisco, 1994.

[435] D. Matzke, "Will Physical Scalability Sabotage Performance Gains?" *IEEE Computer Magazine*, Sept. 1997, Vol.30, No. 9, pp. 37-39.

[436] O. McBryan, "An Overview of Message Passing Environments," *Parallel Computing*, Vol. 20, No. 4, 1994, pp. 417-444.

[437] J. D. McCalpin,"STREAM: Sustainable Memory Bandwidth in Recent and Current High Performance Computers". An on-line technical report, available at http://www.cs.virginia.edu/stream/ or http://reality.sgi.com/mccalpin/papers/bandwidth/bandwidth.html. 1995.

[438] J. D. McCalpin, "Memory Bandwidth and Machine Balance in Current High Performance Computers," *IEEE Technical Committee on Computer Architecture Newsletter*, December 1995.

[439] P. K. McKinley, Y.-J. Tsai, and D.F. Robinson, "Collective Communication in Wormhole-Routed Massively Parallel Computers," *IEEE Computer*, 1995, 28(12): 39-50.

[440] L. McVoy and C. Staelin, "lmbench: Portable Tools for Performance Analysis," *Proc. of the 1996 USENIX Technical Conference*, San Diego, CA, January 1996, pp. 279-295.

[441] J. M. Mellor-Crummey and M. L. Scott, "Algorithms for Scalable Synchronization on Shared Memory Multiprocessors", *ACM Trans. Computer Systems*, 1991, 9(1): 21-65.

[442] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, Version 1.1, June 12, 1995, ftp://ftp.mcs.anl.gov/pub/mpi/mpi-1.jun95/mpi-report.ps, August 1996.

[443] P. Messina and T. Sterling (Eds.), *System Software and Tools for High Performance Computing Environment*, SIAM, Philadelphia, 1993.

[444] R. Metcalfe and D. Boggs, "Ethernet: Distributed Packet Switching for Local Area Networks," *Comm. of ACM*, Vol. 19, No. 7, 1976, pp. 711-719.

[445] MHPCC, "MHPCC 400-Node SP2 Environment", Maui High-Performance Computing Center, Maui, HI, August 1995 (http://www.mhpcc.edu).

[446] Microsoft, *Wolfpack for Windows NT Servers*, Web: http://www.microsoft.com, 1997.

[447] E. Miller and R. Katz, "Input/Output Behavior of Supercomputing Applications," *Proc. Supercomputing '91*, pages 567-576. ACM, 1991.

[448] H.D. Mills, "Structured Programming: Retrospect and Prospect," *IEEE Software*, 1986, 3(6):58-66.

[449] R. Milner, *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, N.J., 1989.

[450] S. Mirapuri, M. Woodacre, and N. Vasseghi, "The MIPS R4000 Processor," *IEEE Micro*, Vol. 12, No. 2, April 1992, pp. 10-22.

[451] G. E. Moore, "Can Moore's Law Continue Indefinitely?," *Computerworld*, July 1996.

[452] C. Morin, A. Gefflaut, M. Banatre, and A.-M. Kermarrec, "COMA: An Opportunity for Building Fault-Tolerant Scalable Shared Memory Multiprocessors," *Proc. of the 23rd Int'l. Symp. on Computer Architecture*, 1996, pp. 56-65.

[453] J. H. Morris, M. Satyanarayanan, M.H. Conner, J.H. Howard, and D.S.H. Rosenthal, "Andrew: A Distributed Personal Environment," *Comm. ACM*, Vol. 29, No. 3, 1986, pp. 184-201.

[454] T. Mowry and A. Gupta, "Tolerating Latency through Software-Controlled Prefetching in Shared-Memory Multiprocessors," *Journal of Parallel and Distributed Computing*, June 1991, pp. 87-106.

[455] MPI Forum, "MPI: A Message-Passing Interface Standard," *Int'l. Journal of Supercomputer Applications*, Vol.8, March 1994.

[456] S. S. Muchnick, "Optimizing Compilers for SPARC," *Sun Technology*, Summer, 1988, pp. 64-77.

[457] T. N. Mudge, J. P. Hayes, and D. D. Winsor, "Multtiple Bus Architectures", *IEEE Computer*, Vol. 20, No. 6, 1987, pp. 42-49.

[458] S. S. Mukherjee, B. Falsafi, M.D. Hill, and D.A. Wood, "Coherent Network Interfaces for Fine-Grain Communication," *Proc. of the 23rd Int'l. Symp. on Computer Architecture*, 1996, pp. 247-257.

[459] H. S. Murayama et al., "A Study of High-Performance Communication Mechanism for Multicomputer Systems," *10th Int'l. Parallel Processing Symp.*, Honolulu, Hawaii, April, 1996, pp. 76-83.

[460] R. Nelson, D. Towsley, and A.N. Tantawi, "Performance Analysis of Parallel Processing Systems," *IEEE Trans. Software Engineering*, April 1988, pp. 532-540.

[461] R. Netzer and J. Xu, "Necessary and Sufficient Conditions for Consistent Global Snapshot," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 6, No. 2, 1994, pp. 165-169.

[462] N. Nevin, "The Performance of LAM 6.0 and MPICH 1.0.12 on a Workstation Cluster," *Technical Report*, OSC-TR-1996-4, Ohio Supercomputer Center, 1996.

[463] L. M. Ni and K. Hwang, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," *IEEE Trans. on Software Engineering*, May 1985, pp. 491-496.

[464] L. M. Ni, "A Layered Classification of Parallel Computers," *Proc. 1991 Int'l. Conf. for Young Computer Scientists*, Beijing, China, May 1991, pp. 28-33.

[465] L. M. Ni and P. K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *IEEE Computer*, Vol. 26, No. 2, Feb. 1993, pp. 62-76.

[466] A. Nicolau and J. A. Fisher, "Measuring the Parallelism Available for Very Long Instruction Word Architectures," *IEEE Trans. on Computers*, Nov. 1984, pp. 968-976.

[467] R. S. Nikhil, *Tutorial Notes on Multithreaded Architectutre*, presented at *the 19th Annual Int'l. Symp. Computer Architecture (ISCA)*, Austrialia, 1992. Contact DEC Cambridge Laboratory, Bldg. 700, 1 Kendall Square, Cambridge, MA.

[468] B. Nitzberg and V. Lo, "Distributed Shared Memory: A Survey of Issues and Algorithms," *IEEE Computer*, Vol. 24, No. 8, August 1991, pp. 52-60.

[469] NOW Project, "Efficient and Portable Implementation of MPI using Active Messages," University of California, Berkeley (http://now.cs.berkeley.edu/Fastcomm/mpi.html), July 10, 1996.

[470] A. G. Nowatzyk, M.C. Browne, E. J. Kelly, and M. Parking, "S-Connect: From Networks of Workstations to Supercomputer Performance," *Proc. of the 22nd Annual Int'l. Symp.on Computer Architecture*, 1995, pp. 71-82.

[471] NSF, "Grand Challenges: High Performance Computing and Communications," A Report by the Committee on Physical, Mathematical and Engineering Sciences, NSF/CISE, 1800 G Street NW, Washington, DC 20550, 1991.

[472] N. Nupairoj and L. M, Ni, "Performance Evaluation of Some MPI Implementations on Workstation Clusters," *Proc. of the Scalable Parallel Libraries Conf.*, Oct. 1994.

[473] D. Nussbaum and A. Agarwal, "Scalability of Parallel Machines," *Comm. of the ACM*, 34(3): 56-61, March 1991.

[474] Open Software Foundation, *OSF DCE Application Environment Specification*, Prentice-Hall, Englewood Cliffs, N.J., 1992.

[475] OpenMP Standards Board, *OpenMP: A Proposed Industry Standard API for Shared Memory Programming*, October 1997, http://www.openmp.org/openmp/mp-documents/paper/paper.html

[476] OpenMP Standards Board, *OpenMP Fortran Application Program Interface Version 1.0*, October 1997, http://www.openmp.org/openmp/mp-documents/fspec.pdf

[477] Oracle Corporation, *Oracle Parallel Server in the Digital Environment*, Oracle Corporation White Paper, Part A19242, June 1994.

[478] J. K. Ousterhourt et al., "The Sprite Network Operating System," *IEEE Computer*, Vol. 21, No. 2, Feb. 1988, pp. 23-26.

[479] D. A. Padua, D. J. Kuck, and D. H. Lawrie, "High-Speed Multiprocessors and Compilation Techniques," *IEEE Trans. on Computers*, pp. 763-776, Sept. 1980.

[480] S. Pakin, V. Karamcheti, and A. Chien, "Fast Messages: Efficient Portable Communication for Workstation Clusters and Massively-Parallel Processors," *IEEE Concurrency*, Vol. 5, No. 2, 1997, pp. 60-73.

[481] C. M. Pancake, "Software Support for Parallel Computing: Where Are We Headed?" *Comm. of the ACM*, Vol. 34, No. 11, 1991, pp. 53-64.

[482] C. M. Pancake and D. Bergmark, "Do Parallel Languages Respond to the Needs of Scientific Programmers?" *IEEE Computer*, Vol. 23, Dec. 1990, pp. 13-23.

[483] C. M. Pancake, M.L. Simmns, and J.C. Yan, "Performance Evaluation Tools for Parallel and Distributed Systems," *IEEE Computer*, Vol. 28, Nov. 1995, pp. 16-19.

[484] D. K. Panda and K. Hwang, "Fast Data Multipulation in Multiprocessors Using Parallel Pipelined Memories," *J. Parallel and Distributed. Computing*, Vol. 12, June 1991, pp. 130-145.

[485] D. K. Panda and C. B. Stunkel (Eds.) *Communication and Architectural Support for Network-Based Parallel Computing*, Spinger-Verlag, Heidelberg, Germany, 1997.

[486] M. S. Papamaroos and J.H. Patel, "A Low-Overhead Coherence Solution for Multiprocessor with Private Cache Memories", *Proc. of the 11th Annual Symp. Computer Achitecure*, June 1984, pp. 348-354.

[487] D. B. Papworth, "Tuning the Pentium Pro Microarchitecture," *IEEE Micro*, Vol. 16, No. 2, 1996, pp. 8-15.

[488] Parallel Computing Forum, "PCF: Parallel Fortran Extensions," *Fortran Forum*, Vol. 10, No. 3, September 1991.

[489] C. Partridge, *Gigabit Networking*, Addison-Wesley, Reading, Mass., 1994.

[490] D. M. Pase, T. MacDonald, and A. Meltzer, "The CRAFT Fortran Programming Model," *Scientific Programming*, Vol. 3, 1994, pp. 227-253.

[491] Y. N. Patt, S. J. Patel, M. Evers, D. H. Friendly, and J. Stark, "One Billion Transistors, One Uniprocessor, One Chip," *IEEE Computer Magazine*, Vol.30, No. 9, Sept. 1997, pp. 51-57

[492] D. Patterson and J.L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, Morgan Kaufmann, San Francisco, 1994.

[493] D. Patterson and C. Sequin, "A VLSI RISC," *IEEE Computer*, Vol. 15, Sept. 1982.

[494] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz, "NFS Version 3 Design and Implementation," *Proc. of the 1994 Summer USENIX Conference*, USENIX Association, Berkeley, CA, June 1994.

[495] A. Peleg, "MMX Technology Extension to the Intel Architecture," *IEEE Micro*, Vol. 16, No. 4, 1996, pp. 42-50.

[496] S. Petri and H. Langendorfer, "Load Balancing and Fault Tolerance in Workstation Clusters Migrating Groups of Communicating Processes," *Operating Systems Review*, Vol. 29, No. 4, Oct. 1995, pp. 25-36.

[497] G. F. Pfister, *In Search of Clusters*, Prentice-Hall PTR, Upper Saddle River, NJ, 1995.

[498] G. F. Pfister, "Clusters of Computers: Characteristics of an Invisible Architecture," Keynote address presented at *IEEE Int'l. Parallel Processing Symp.*, Honolulu, April 1996.

[499] G. F. Pfister and V.A. Norton, "Hot Spot Contention and Combining in Multistage Interconnection Networks," *Proc. Int''l. Conf. Parallel Processing*, 1995, pp. 790-797.

[500] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norlton, and J. Weiss, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," *Proc. Intl Conf. on Parallel Processing*, 1985, pp. 764-771.

[501] J.A. Piantenosi, A. S. Sathaye, and D. J. Shakshober, "Performance Measurement of TruCluster Systems under the TPC-C Benchmark," *Digital Technical Journal*, Vol. 8, No. 3, 1996, pp. 46-57.

[502] P. Pierce and G. Regnier, "The Paragon Implementation of the NX Message Passing Interface," *Proc. of the Scalable High-Performance Computing Conference*, May 1994, pp. 184-190.

[503] J. S. Plank, M. Beck, G. Kingsley, and K. Li, "Libckpt: Transparent Checkpointing under Unix," *Proc. of 1995 Winter USENIX Technical Conference*, pp. 213-224.

[504] J. S. Plank and K. Li, "Faster Checkpointing with *N*+1 Parity," *Proc. of the 24th Int'l. Symp. on Fault Tolerant Computing*, 1994, pp. 288-297.

[505] J. S. Plank, J. Xu, and R. Netzer, "Compressed Differences: An Algorithm for Fast Incremental Checkpointing," *Tech. Report CS-95-302*, University of Tennessee, 1995.

[506] R. Ponnusamy, J. Saltz, A. Choudhary, Y.-S. Hwang, and G. Fox, "Runtime Support and Compilation Methods for User-Specified Data Distributions," *IEEE Trans. on Parallel and Distributed Systems*, August, 1995.

[507] R. Ponnusamy, J. Saltz, A. Choudhary, Y.-S. Hwang, and G. Fox, "Supporting Irregular Data Distributions in FORTRAN 90D/HPF Compilers," *IEEE Parallel and Distributed Technology*, Spring 1995.

[508] F. P. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *Comm. of the ACM*, Vol. 24, 1981, pp. 300-309.

[509] J. Protic, M. Tomaševic and V. Milutinovic (Eds.), *Distributed Shared Memory: Concepts and Systems*, IEEE Computer Society Press, August 1997.

[510] S. Przybylski, *Cache and Memory Hierarchy Design*, Morgan Kaufmann, San Mateo, CA, 1990.

[511] M. Quinn, *Parallel Computing: Theory and Practice*, McGraw-Hill, New York, 1994.

[512] M. J. Quinn and P. J. Hatcher, "Data-Parallel Programming on Multicomputers," *IEEE Software*, 7(5): 69-76, Sept. 1990.

[513] J. Rattner, "Desktops and TeraFLOP: A New Mainstream for Scalable Computing," *IEEE Parallel and Distributed Technology*, August 1993, pp. 5-6.

[514] D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. W. Schwartz, and L. F. Tavera, "Scalable Performance Analysis: The Pablo Performance Analysis Environment," *Proc. Scalable Parallel Libraries Conf.*, 1993, pp. 104-113.

[515] S. K. Reinhardt, M.D. Hill, J.R. Larus, A.R. Lebeck, J.C. Lewis, and D.A. Wood,"The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers," *Proc. of the 1993 ACM SIGMETRICS Conf.* 1993, pp. 48-60.

[516] S. K. Reinhardt, R.W. Pfile, and D.A. Wood, "Decoupled Hardware Support for Distributed Shared Memory," *Proc. of the 23rd Int'l. Symp. on Computer Architecture*, 1996.

[517] M.C. Rinard, D.J. Scales, and M.S. Lam, "Jade: A High-Level, Machine-Independent Language for Parallel Programming," *IEEE Computer*, Vol. 26, June 1993, pp. 28-38.

[518] E. Roberts, *"Gigabit Ethernet: Weighed Down by Doubts," Data Communications*, November 1996.

[519] M. Rosenblum, E. Bugnion, S.A. Herrod, E. Witchel, and A. Gupta, "The Impact of Architectural Trends on Operating System Performance," *Proc. of the 15th ACM Symp. on Operating Systems Principles*, December 1995, pp. 285-298.

[520] M. Rosenblum, J. Chapin, D. Teosdosiu, S. Devine, T. Lahiri, and A. Gupta, "Implementing Efficient Fault Containment for Multiprocessors". *Comm. of the ACM*, Vol. 39, Sept. 1996, pp. 52-61.

[521] R. H. Saavedra, D. E. Culler, and T. von Eicken, "Analysis of Multithreaded Architecture for Parallel Computing," *Proc. of ACM Symp. Parallel Algorithms and Architecture*, Greece, July 1990.

[522] R .H. Saavedra, W. Mao and K. Hwang, "Performance and Optimization of Data Prefetching Strategies in Multiprocessors," *Journal of Parallel and Distributed Computing*, Sept. 1993, pp. 427-448.

[523] R. H. Saavedra and A.J. Smith, "Analysis of Benchmark Characteristics and Benchmark Performance Prediction," *ACM Trans. on Computer Systems*, Vol. 14, No. 4, 1996, pp. 344-384.

[524] S. Sahni and V. Thanvantri, "Performance Metrics: Keeping the Focus on Runtime," *IEEE Parallel and Distributed Technology*, Spring 1996, pp. 43-56.

[525] S. Saini and D.H. Bailey, "NAS Parallel Benchmark Results 12-95," *Technical Report* NAS-95-021, NASA Ames Research Center, Dec. 1995.

[526] M. Santayaraman, "Scalable, Secure, and Highly Aavailable Distributed File Access," *IEEE Computer*, Vol. 23(5), May 1990, pp. 9-21.

[527] R. Sandberg, D. Goldberg, S.R. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network Filesystem," *Proc. of the Summer 1985 USENIX Technical Conference*, 1985, pp. 119-130.

[528] W. Saphire, L.A. Tanner, and B. Traversat, "Job Management Requirements for NAS Parallel Systems and Clusters," *Technical Report NAS-95-006*, NASA Ames Research Center, 1995.

[529] W. Saphire, A. Woo, and M. Yarrow; "The NAS Parallel Benchmarks 2.1 Results," NASA Ames Research Center, *Technical Report NAS-96-010*, NASA Ames Research Center, August, 1996.

[530] S. Saunders (Ed.), *The McGraw-Hill High-Speed LANs Handbook*, McGraw-Hill, New York, 1996.

[531] J. Singh, W.-D. Weber, and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared Memroy," ACM SIGARCH *Computer Architecture News* Vol. 20(1), March 1992, pp. 5-44.

[532] R. Sites, "Alpha AXP Architecture," *Comm. of the ACM*, Vol. 36, No. 2, Feb. 1993, pp. 33-44.

[533] J. E. Smith and S. Vajapeyam, "Trace Processors Moving to Fourth Generation Microarchitectures," *IEEE Computer*, Vol.30, No. 9, Sept. 1997, pp. 68-74.

[534] M. Syanarayanan, "Commercial Multiprocessign Systems," *IEEE Computer*, May 1980, pp. 75-96.

[535] K .E. Schauser and C.J. Scheiman, "Active Messages Implementations for Meiko CS-2," Department of Computer Science, University of California, Santa Barbara, 1994.

[536] C. Scheurich, *Access Ordering and Coherence in Shared-Memory Multiprocessors*, Ph.D. Thesis, University of Southern California, 1989.

[537] C. Schimmel, *UNIX Systems for Modern Architectures: Symmetric Multiprocessing and Caching for Kernel Programmers*, Addison-Wesley Pub. Co., Menlo Park, CA. 1994.

[538] J. T. Schwartz, "Ultra-Computers," *ACM Trans. Prog. Lang. and Systems*, Vol. 2, April, 1980, pp. 484-521.

[539] S. L. Scott, "The GigaRing Channel," *IEEE Micro*, Feb. 1996, pp. 27-34.

[540] S. L. Scott, "Synchronization and Communication in the T3E Multiprocessor," *Proc. of ASPLOS 7*, Oct. 1996, pp. 26-36.

[541] S. L. Scott and G. Thorson, "The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus," *Hot Interconnects IV*, August, 1996.

[542] SDSC, "SDSC's Intel Paragon," San Diego Supercomputer Center, http://www.sdsc.edu/Services/Consult/Paragon/paragon.html.

[543] C. L. Seitz, "The Cosmic Cube," *Comm. of the ACM*, Vol. 28, Jan. 1985, pp. 22-33.

[544] C. L. Seitz, J. Seizovic, and W. K. Su, "The C-Programmer's Guide to Multicomputer Programming," *Technical Report CS-TR-88-1*, California Institute of Technology, Pasadena, CA, 1989.

[545] C. L. Seitz, Concurrent Architectures," in Suaya and Birtwistle (eds.), *VLSI and Parallel Computation*, Chapter 3, Morgan Kaufmann, San Mateo, CA, 1990.

[546] C. L. Seitz, "Mosaic C: An Experimental Fine-Grain Multicomputer," *Technical Report*, California Institute of Technology, Pasadena, CA, 1992.

[547] Silicon Graphics, *IRIS Power C User's Guide*, Silicon Graphics Computer Systems, Mounbtain View, CA. 1989.

[548] Silicon Graphics, *POWER CHALLENGEarray Technical Report*, Silicon Graphics Computer Systems, Mounbtain View, CA. 1996.

[549] Silicon Graphics, *POWER CHALLENGE Technical Report*, Silicon Graphics Computer Systems, Mountain View, CA. 1996.

[550] Silicon Graphics, *Origin 200 and Origin 2000 Technical Report*, Silicon Graphics Computer Systems, Mountain View, CA. 1997.

[551] J. Shah, *VAXclusters*, McGraw-Hill, New York, 1991.

[552] S. S. Shang and K. Hwang, "Distributed Hardwired Barrier Synchronization for Scalable Multiprocessor Clusters," *IEEE Trans. Parallel and Distributed Computing Systems*, June 1996, pp. 591-605.

[553] N. Shavit and D. Touitou, "Software Transactional Memory," *Distributed Computing*, Vol. 10, No. 2, 1997, pp. 99-116.

[554] M. S. Shur, S.M. Sze, and J.M. Xu (Eds.), "Special Issue on Present and Future Trends in Device Science and Technology," *IEEE Trans. on Electronic Devices*, Vol. 43, No. 10, 1996.

[555] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*, McGraw-Hill, New York, 1989.

[556] H. J. Siegel et al., "Report of the Purdue Workshop on Grand Challenges in Computer Architecture for the Support of High Performance Computing," *J. Parallel and Distributed Computing*, Vol. 16, No. 3, 1992, pp. 199-211.

[557] D. P. Siewiorek and P. J. Koopman, *The Architecture for Supercomputers, TI-TAN: A Case Study*, Academic Press, New York, 1991.

[558] P. S. Sindhu, J. M. Frailong, and M. Cekleov, "Formal Specification of Memory Modules," in *Scalable Shared-Memory Multiprocessors*, Kluwer Academic Publisheds, Boston, MA, 1992.

[559] J. Singh, J. L. Hennessy, and A. Gupta, "Scaling Parallel Programs for Multiprocessors: Methodology and Examples," *IEEE Computer*, Vol. 26, No. 7, 1993, pp. 42-50.

[560] J. Singh, J. L. Hennessy, and A. Gupta, "Implication of Hierachical N-Body Methods for Multiprocessor Architectures," *ACM Trans. on Computer Systems*, Vol. 13, Feb. 1995, pp. 141-202.

[561] A. Singhal, D. Broniarczyk, F. Cerauskis, J. Price, L. Yuan, C. Cheng, D. Doblar, S. Fosth, N. Agarwal, K. Harvey and E. Hagersten, "Gigaplane: A High Performance Bus for Large SMPs," *Symposium Record of Hot Interconnects IV*, August 1996.

[562] M. Singhal, "Deadlock Detection in Distributed Systems," *IEEE Computer*, Nov. 1989, pp. 37-48.

[563] R L. Sites and R. T. Witek (Eds.), *Alpha AXP Architecture Reference Manual*, 2d ed., Digital Press, Boston, MA. 1995.

[564] A. Skjellum, "The Multicomputer Toolbox: Current and Future Directions," *Proc. 1993 IEEE Scalable Parallel Libraries Conf.*, pp. 94-103.

[565] A. Skjellum, P. Vaughan, and C. Roberts, "UNIFY: Interoperable MPI and PVM Programming in a Workstation-Network Environment," *MPI Developers Conference*, University of Notre Dame, June 1995.

[566] M. Slater, "The Microprocessor Today," *IEEE Micro*, Vol. 16, No. 6, 1996, pp. 32-45.

[567] M. D. Smith, M. Johnson, and M. A. Horowitz, "Limits on Multiple Instruction Issue," *Proc. of the Third Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems*, April 1989, pp. 290-302.

[568] A. J. Smith, "Cache Memories," *ACM Computing Survey*, pp. 473-530, Sept. 1982.

[569] B. J. Smith, "The Quest for General-Purpose Parallel Computing," Tera Computer Company, 1990, http://www.tera.com.

[570] B. J. Smith, "The Architecture of HEP," *Parallel MIMD Computation: The HEP Supercomputer and its Applications*, edited by J. S. Kowalik, MIT Press, 1985, pp. 41-55.

[571] J. E. Smith, "Using Standard Microprocessors in MPPs," presented at the *Int'l. Symp.on Computer Architecture*, 1992.

[572] J. E. Smith and S. Vajapeyam, "Trace Processor: Moving to Fourth-Generation Microarchitectures," *IEEE Computer*, Sept. 1997, pp. 68-74.

[573] M. Smith, "Closing on Clusters: Will Wolfpack Dominate the High-Volume Windows NT Cluster Market?" *Windows NT Magazine*, Aug. 1996, also from http://www.winntmag.com/issues/Aug96/wolfpack.htm, July 14, 1997.

[574] M. Snir, P. Hochschild, D.D. Frye, and K.J. Gildea, "The Communication Software and Parallel Environment of the IBM SP2," *IBM Systems Journal*, Vol. 34, No. 2, 1995, pp. 205-221.

[575] M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, and J. Dongarra, *MPI: The Compelete Reference*, The MIT Press, Combridge, Mass., 1996.

[576] L. Snyder. "Type Architectures, Shared Memory, and the Corollary of Modest Potential," *Annual. Review of. Comput. Science*, Vol. 1, 1986, pp. 289-317.

[577] G. Sohi, S. Breach, and T. N. Vijaykumar, "Multiscalar Processors," *Proc. of the 22nd Annual International Symposium on Computer Architecture*, June 1995, pp. 414-425.

[578] S. Song, M. Denman, and J. Chang, "The PowerPC 604 RISC Microprocessor," *IEEE Micro*, Oct. 1994, pp. 8-17.

[579] W. Stallings (Ed.), *Advances in Local and Metropolitan Networks*, IEEE Computer Society Press, 1994.

[580] W. Stallings (Ed.), *Advances in ISDN and Broadband ISDN*, IEEE Computer Society Press, 1992.

[581] W. Stallings, *Operating Systems* (2nd Ed.), Prentice-Hall, Englewood Cliffs, NJ., 1995.

[582] D.A. Stamper, *FDDI Handbook: High-Speed Networking Using Fiber and Other Media*, Addison Wesley, Reading Mass., 1994.

[583] D. Stein and D. Shah, "Implementing Lightweight Threads," *Proc. of the USENIX Summer Conference*, 1992.

[584] P. Stenstrom, "A Survey of Cache Coherence Schemes for Multiprocessors," *IEEE Computer* 23(6):12-24, June 1990.

[585] P. Stenstrom et al., "Comparative Performance Evaluation of Cache Coherent NUMA and COMA Architectures," *Proc. of the 19th Annual Int'l. Symp. on Computetr Architecture*, 1992, pp. 80-91.

[586] T. Sterling, P. Merkey, and D. Savarese, "Improving Application Performance on the HP/Convex Exemplar," *IEEE Computer*, Vol. 29, No. 12, Dec. 1996, pp. 50-55.

[587] T. Sterling, P. Messina, and P. H. Smith, *Enabling Technologies for Petaflops Computing*, MIT Press, Cambridge, Mass., 1995.

[588] H. S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Trans. Computers*, Vol. 20, 1971.

[589] J. M. Stone, H. S. Stone, P. Heidelberger, and J. Turek, "Multiple Reservations and the Oklahoma Update," *IEEE Parallel and Distributed Technology*, Spring 1993, 1(4): 58-71.

[590] H. S. Stone, *High-Performance Computer Architectures (Third Edition)*, Addison-Wesley, 1993.

[591] T. Stricker and T. Gross, "Optimizing Memory System Performance for Communication in Parallel Computers," *Proc. of the 22nd Annual Int'l. Symp. on Computer Architecture*, 1995, pp. 308-319.

[592] C. B. Stunkel, D. G. Shea, B. Abali, M. G. Atkins, C. A. Bender, D.G. Grice, P. Hochschild. D. J. Joseph, B. J. Nathanson, R. A. Swetz, R. F. Stucke, M. Tsao, and P. R. Varker, "The SP2 High-Performance Switch," *IBM Systems Journal*, 34(2):185-204, 1995.

[593] Y. Sun, J. Wang, and Z. Xu, "Architectural Implications of the NAS MG and FT Parallel Benchmarks," *Proc. of the Int'l. Conference on Advances in Parallel and Distributed Computing*, March 1997, pp. 235-240.

[594] X. H. Sun and D. Rover, "Scalability of Parallel Algorithm-Machine Combinations," *IEEE Trans. on Parallel and Distributed Systems*, May, 1994.

[595] X. H. Sun and J. Zhu, "Performance Considerations of Shared Virtual Memory Machines," *IEEE Trans. on Parallel and Distributed Systems*, Nov. 1995.

[596] X. H. Sun, and L. Ni, "Scalable Problems and Memory-Bounded Speedup," *Journal of Parallel and Distributed Computing*, Vol. 19, Sept. 1993, pp. 27-37.

[597] V. S. Sunderram, "PVM: A Framework for Parallel Distributed Computing," *Concurrency: Practices and Experience*, Dec. 1990, pp. 315-339.

[598] Sun, *SPARC Architecture Reference Manual V8*, Sun Microsystems, Inc., CA. Dec. 1990.

[599] Sun, *SPARC-Cluster 1 Product Overview*, Sun Microsystems, Mountain View, CA. 1994.

[600] Sun, *The Ultra Enterprise 10000 Server: Technical White Paper*, Sun Microsystems, Mountain View, CA. 1997. Available from http://www.sun.com/.

[601] SunSoft, *Multithreaded Programming Guide*, Sun Microsystems, 1994.

[602] D. Tabak, *Advanced Microprocessors*, McGraw-Hill, New York, 1995.

[603] D. Tabak, *RISC Systems and Applications*, Research Studies Press, Wiley, New York, 1996.

[604] R. Take, "A Routing Method for the All-to-All Burst on Hypercube Network," *Proc. 35th National Conf. of Information Processing Society of Japan*, 1987, pp. 151-152.

[605] Tandem Computers Incorporated. *NonStop Himalaya Range*. Document # CD0194-0993.

[606] C. K. Tang, "Cache Design in the Tightly Coupled Multiprocessor System", *Proc. AFIPS National Computer Conf.*, New York, June 1976, pp. 749-753.

[607] A. S. Tannenbaum, *Distributed Operating Systems*, Prentice-Hall, N.J., 1995.

[608] T. Tannenbaum and M. Litzkow, "The Condor Distributed Processing System," *Dr. Dobb's Journal*, Feb. 1995, pp. 40-48.

[609] D. Teodosiu, J. Baxter, K. Govil, J. Chapin, M. Rosenblum, and M. Horowitz, "Hardware Fault Containment in Scalable Shared-Memory Multiprocessors," *Proc. of the 24th Annual Int'l. Symp. on Computer Architecture*, 1997, pp. 73-84.

[610] S. S. Thakkar, M. Dubois, A. T. Laundrie, G. S. Sohi, D. V. James, S. Gjessing, M. Thapar, B. Delagi, M. Carlton, and A. Despain, "New Directions in Scalable Shared-Memory Multiprocessor Architectures," *IEEE Computer*, Vol. 23, No. 6, 1990, pp. 71-83.

[611] R. Thekkath and S.J. Eggers, "The Effectiveness of Multiple Hardware Contexts," *Proc. 6th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 1994, pp. 328-337.

[612] TMC, *The CM-5 Technical Summary*, Thinking Machines Corporation, Cambridge, MA, 1995.

[613] D. Tolmie and D. Flanagan, "HIPPI: It's Not Just for Supercomptuers Anymore," *Data Communciatiuons*, http://www.data.com/

[614] R. M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM J. Res. and Develop.*, 11(1):25-33, 1997.

[615] H. C. Torng and S. Vassiliadis, *Instruction-Level Parallel Processors*, IEEE Computer Society Press, Los Alamitos, CA. 1995.

[616] J. Torrellas, M. S. Lam, and J. L. Hennessy, "False Sharing and Spatial Locality in Multiprocessor Caches," *IEEE Trans. on Computers*, June 1994, Vol. 43, No. 6, pp. 651-663.

[617] M. Tremblay and J.M. O'Connor, "UltraSparc I: A Four-Issue Processor Supporting Multimedia," *IEEE Micro*, Vol. 16, No. 2, 1996, pp. 42-50.

[618] L. W. Tucker and A. Mainwaring, "CMMD: Active Messages on the CM-5," *Parallel Computing*, Vol. 20, 1994, pp. 481-496.

[619] D. M. Tullsen and S. J. Eggers, "Effective Cache Prefetching on Bus Based Multiprocessors," *ACM Trans. on Computer Systems*, Vol.3, Jan. 1995, pp. 57-88.

[620] L. H. Turcotte, "A Survey of Software Environments for Exploiting Networked Computing Resources," *Technical Report* MSU-EIRS-ERC-93-2, Mississippi State University, 1993.

[621] U. Vahalia, *Unix Internals: The New Frontiers*, Prentice-Hall, Englewood Cliffs, NJ., 1996.

[622] A. Vahdat, D. Ghormley, and T. Anderson, "Efficient, Portable, and Robust Extension of Operating System Functionality," *Technical Report*, Computer Science Division, UC Berkeley, Dec. 1994.

[623] L. G. Valiant, "A Bridging Model for Parallel Computation," *Comm. of ACM*, 33(8):103-111, 1990.

[624] A. Varma and C. Raghavendra, *Interconnection Networks for Multiprocessors and Multicomputers: Theory and Practice*. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[625] E. F. Van De Velde, *Concurrent Scientific Computing*, Springer-Verlag, 1994.

[626] T. von Eiken, A. Basu, and V. Buch, "Low-Latency Communication over ATM Network Using Active Messages," *IEEE Micro*, Feb. 1995, pp. 46-53.

[627] T. von Eiken, A. Basu, V. Buch, and W. Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing," *Proc. of 15th ACM Symp. on Operating Systems Principles*, 1995.

[628] T. von Eiken, D.E. Culler, S.G. Goldstein, and K.E. Schauser, "Active Messages, A Mechanism for Integrated Communication and Computation," *Proc. of the 19th Annual Intl. Symp. on Computer Architecture*, 1992, pp. 256-266.

[629] B. W. Wah and C. V. Ramamoorthy (Eds.), *Computers for Artificial Intelligence Processing*, Wiley, New York, 1990.

[630] R. Wahbe, S. Lucco, and T. Anderson, "Efficient Software-Based Fault Isolation," *Proc. 14th ACM Symp. on Operating Systems Principles*, Dec. 1993, pp. 203-216.

[631] E. Waingold, M. Taylor, D. Srikrishna, and V. Sarkar, "Baring It All to Software: Raw Machines," *IEEE Computer Magazine*, Vol.30, No. 9, Sept. 1997, pp. 86-93.

[632] D. Wall, "Limits in Instruction-Level Parallelism," *Proc. Arch. Support for Prog. Languages and Operating Systems*, April 1991, pp. 176-188.

[633] B. J. Walker, J. Lilienkamp, and J. Hopfield, *Open Single System Image Software for Multicomputer or MPP (Massively Parallel Processor)*, Locus Computing Corporation, 9800 La Cienega Blvd., Ingelwood CA 90301-4400, 1993.

[634] C. J. Wang, C.-L. Wang, and K. Hwang, "STAP Benchmark Evaluation of the T3D, SP2, and Paragon," *Proc. of 10th Int'l. Conf. on Parallel and Distributed Computing Systems*, New Orleans, Oct. 1-3, 1997.

[635] H. C. Wang and K. Hwang, "Multicoloring for Solving Multigrid PDE Problems on Shared-Memory Multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, Nov. 1996, pp. 1195-1205.

[636] T. M. Warschko, J. M. Blum, and W. F. Tichy, "The ParaStation Project: Using Workstations as Building Blocks for Parallel Computing," *Proc. of Int'l. Conf. on Parallel and Distributed Processing, Techniques and Applications (PDPTA'96)*, August 1996, Sunnyvale, CA, Vol I, pp. 375-386.

[637] D. L. Weaver and T. Germond (Eds.), *SPARC Architecture Manual*, Prentice-Hall, Englewood Cliffs, N.J., 1994.

[638] W. -D. Weber, *Scalable Directories for Cache-Coherent Shared-Memroy Multiprocessors*, Ph.D. Thesis, Stanford University, January 1993.

[639] W. -D. Weber et al., "The Mercury Interconnect Architecture: A Cost-Effective Infrastructure for High-Performance Servers," *Proc. of the 24th Annual Intl. Symp. on Computer Architecture*, 1997, pp. 98-107.

[640] U.Weiser, "Intel MMX Technology - An Overview," *Proc. Hot Chips Symp.*, Aug. 1996, p.142.

[641] M. Welsh, A. Basu, and T. von Eicken, "ATM and Fast Ethernet Network Interfaces for User-level Communication," *Proc. of the Third International Symposium on High Performance Computer Architecture (HPCA)*, San Antonio, Texas, 1997.

[642] S. R. Wheat, R. Riesen, A.B. Maccabe, D.W. van Dresser, and T.M. Stallcup, "PUMA: An Operating System for Massively Parallel Systems," *Proc. 27th Hawaii Int'l. Conf. on Systems Sciences*, Vol. II, 1994, pp. 56-64.

[643] S. Whitney et al., "The SGI Origin Software Environment and Application Performance," *Proc. of COMPCON Spring 97*,IEEE Computer Society, Feb, 1997, pp. 165-170.

[644] A. W. Wilson, "Hierarchical Cache/Bus Architecture for Shared-Memory Multiprocessors," *Proc. 14th Annu. Int. Symp. Computer Arch.*, pp. 244-252, 1987.

[645] G. V. Wilson and P. Lu (Eds.), *Parallel Programming Using C++*, The MIT Press, Cambridge, Massachusetts, 1996.

[646] I. Wladawsky-Berger, "The Power and Promise of Parallel Computing," *IBM System Journal*, 1995, 34(2):146-151.

[647] M. Wolfe, *High-Performance Compilers for Parallel Computing*, Addison-Wesley, Redwood City, CA, 1996.

[648] K. F. Wong and M. Franklin, "Checkpointing in Distributed Computing Systems," *Journal of Parallel and Distributed Computing*, Vol. 35, 1996, pp. 67-75.

[649] S. C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Proc. of the 22nd Annual Intl. Symp. on Computer Architecture*, 1995, pp. 24-36.

[650] P. H. Worley, "Limits on Parallelism in the Numerical Solution of Linear PDEs," *SIAM J. Sci. and Stat. Computing*, Vol. 12, No. 1, 1991, pp. 1-35.

[651] J. Worlton, "Characteristics of High-Performance Computers," *in Supercomputers: Directions in Technology and its Applications*, National Academy Press, 1989, pp. 21-50.

[652] J. Xu and K. Hwang, "Heuristic Methods for Dynamic Load balancing in a Message-Passing Multicomputer," *Journal of Parallel and Distributed Computing*, Vol. 18, No. 1, May 1993, pp. 1-13.

[653] Z. Xu and K. Hwang, "Molecule: A Language Construct for Layered Development of Parallel Programs," *IEEE Trans. on Software Engineering*, Vol. SE-15, No. 5, 1989, pp. 587-599.

[654] Z. Xu and K. Hwang, "Language Constructs for Structured Parallel Programming," *Proc. of the Sixth Int'l. Parallel Processing Symp.*, 1992.

[655] Z. Xu and K. Hwang, "Modeling Communication Overhead: MPI and MPL Performance on the IBM SP2 Multicomputer," *IEEE Parallel and Distributed Technology*, Spring 1996, pp. 9-23.

[656] Z. Xu and K. Hwang, "Early Prediction of MPP Performance: SP2, T3D, Paragon Experiences," *Parallel Computing*, Oct. 1996.

[657] Z. Xu and K. Hwang, "MPP versus Clusters for Scalable Computing," *Proc. of the 2nd Int'l. Symp. on Parallel Architectures, Algorithms, and Networks*, IEEE Computer Society Press, June, 1996, pp. 117-123.

[658] Z. Xu and K. Hwang, "Coherent Parallel Programming in C//," *Proc. of Int'l. Conf. on Advances in Parallel and Distributed Computing*, IEEE Computer Society Press, March 1997, pp. 116-122.

[659] J.-H. Yang and J. Anderson, "A Fast, Scalable Mutual Exclusion Algorithm," *Distributed Computing*, Vol. 9, No. 1, August 1995, pp. 51-60.

[660] K.C. Yeager, "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, Vol. 16, No. 2, 1996, pp. 28-41.

[661] P. C. Yew, N. F. Tseng, and D. Lawrie, "Distributing Hot-Spot Addressing in Large-Scale Multiprocessors," *IEEE Trans. on Computers*, pp. 388-395, April 1987.

[662] P. C. Yew and B. W. Wah (Eds.), Special Issue on Shared-Memory Multiprocessors, *J. Parallel and Distributed Computing*, June 1991.

[663] A. Yu, "The Future of Microprocessors," *IEEE Micro*, Vol. 16, No. 6, 1996, pp. 46-53.

[664] E. W. Zegura, "Architecture for ATM Switched Systems," *IEEE Communications*, Feb. 1993, 28-37.

[665] X. Zhang, R. Castaneda, and E. W. Chan, "Spin-Lock Synchronization on the Butterfly and KSR1," *IEEE Parallel and Distributed Technology*, Spring 1994, pp. 51-63.

[666] X. Zhang, Y. Yan, and R. Castaneda, "Evaluating and Designing Software Mutual Exlusion Algorithms on Shared-Memory Multiprocessors," *IEEE Parallel and Distributed Technology*, Spring 1996, pp. 25-42.

[667] S. Zhou, *LSF: Load Sharing and Batch Queueing Software*, Platform Computing Corporation, North York, Canada, 1996.

[668] S. Zhou, X. Zheng, J. Wang, and P. Delisle, "Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems," *Software – Practice and Experience*, 23(12):1305-1336, December 1993.

[669] H. Zima and B. Chapman, *Supercompilers for Parallel and Vector Computers*, Addison-Wesley, Reading, Mass., 1991.

[670] G. Zorpetta, "The Power of Parallelism," *IEEE Spectrum*, Vol. 29, Sept.1992, pp. 28-33.

[671] R. Zucker and J. L. Baer, "A Performance Study of Memeory Consistency Models," *Proc. 19th Int'l. Symp. on Computer Architecture*, May 1992, pp. 2-12.

색 인

ㄹ

ㅁ

ㅂ

ㅅ

ㅈ

ㅊ

ㅋ

ㅍ

ㅎ

ㅆ

ㅇ